VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE 3D SCÉNY PRO OVLÁDÁNÍ ROBOTA

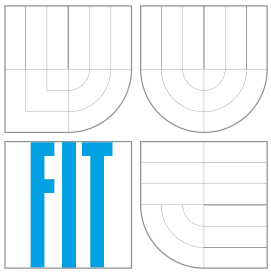
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR BLAHOŽ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZACE 3D SCÉNY PRO OVLÁDÁNÍ ROBOTA

VISUALIZATION ENVIRONMENT FOR ROBOT REMOTE CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR BLAHOŽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2012

Abstrakt

Tato práce se zaměřuje na možnosti využití datové fůze 3D skenu - point cloudu a barevného digitálního videa v procesu vzdáleného ovládání robotů. Diskutují se zde výhody zobrazení okolí robota kombinací dat z více senzorů, prostředky umožňující takovou fůzi a jsou navrženy dvě varianty grafické vizualizace kombinovaných dat z point cloudu a barevného videa. První navržená alternativa se zabývá myšlenkou zobrazení výstupu barevné kamery, a tedy barevné informace o okolí, v prostoru zobrazené 3D scény z dat laserového skeneru na poloprůhledném polygonu umístěném v pohledovém objemu robota. Druhou navrženou možností je přímé obarvování dat 3D skeneru za vzniku barevného point cloudu reprezentujícího barevnou i hloubkovou informaci o okolí.

Abstract

This thesis presents possibilities of 3D point cloud and true colored digital video fusion that can be used in the process of robot teleoperation. Advantages of a 3D environment visualization combining more than one sensor data, tools to facilitate such data fusion, as well as two alternative practical implementations of combined data visualization are discussed. First proposed alternative estimates view frustum of the robot's camera and maps real colored video to a semi-transparent polygon placed in the view frustum. The second option is a direct coloring of the point cloud data creating a colored point cloud representing color as well as depth information about an environment.

Klíčová slova

Datová fůze, point cloud, texturovaný cloud, laserový sken, 3D model

Keywords

Data fusion, point cloud, textured cloud, laser scanning, 3D model

Citace

Vladimír Blahož: Vizualizace 3D scény pro ovládání robota, diplomová práce, Brno, FIT VUT v Brně, 2012

Vizualizace 3D scény pro ovládání robota

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D.

. . . . . .
Vladimír Blahož
16. května 2012

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vzdálené ovládání robota je téma nabízející spoustu otázek pro různá odvětví nejen počítačové grafiky. Samotný pojem *vzdálené ovládání* se týká dvou anglických pojmů *teleoperation* a *remote control*, které v důsledku označují stejnou činnost, přestože výrazu *teleoperation* je častěji používáno ve vědeckém kontextu. Všechny tyto pojmy jsou obvykle používány ve spojení s ovládáním robotů, nicméně principielně se mohou týkat ovládání jakéhokoliv přístroje na libovolnou vzdálenost.

Jde o oblast dynamicky se vyvíjející a zvyšující se nároky na ovládací prostředí pro teleoperátory s sebou přináší veliké zvýšení efektivity práce operátorů, jejich komfortu, bezpečnosti jak pro operátora, tak pro robota a mnoho dalšího. Požadavky na prostředí pro ovládání robotů se týkají především co nejpohodlnějšího přístupu k ovládacím prvkům nebo podoby a rozmístění vizualizačních prvků. Vhodně navržené a rozmístěné prvky prostředí (jak vizualizační, tak ovládací) pro vzdálené ovládání mohou usnadnit práci operátorům, zlepšit jejich pracovní výkonnost, nebo se vyhnout problémům spojeným s dezorientací operátora v prostředí robota, která může vést v nejhorším případě až k poškození robota.

Tato práce pojednává o možnostech vizualizace 3D scény, v níž se robot pohybuje, s ohledem na co nejpřirozenější vjem pro vzdáleného operátora. Jsou zde navrženy možnosti zobrazení prostředí robota za pomoci dat z 3D laserového senzoru a snímků z barevné kamery, a různých způsobů fúze těchto dat. Každá z alernativ vizualizace přináší jiné výhody a nevýhody a jejich využití je především otázkou preference očekávaných vlastností. První navrhovaná alternativa počítá s možností vhodného překrytí vzájemně zarovnaných dat z obou senzorů tak, aby přes trojrozměrná data laserového senzoru byl umístěn odpovídající barevný snímek z kamery. Druhá varianta nabízí možnost doplnění jednotlivých bodů point cloudu o jim příslušející reálnou barvu, odvozenou z barevného snímku kamery. Od obou vizualizací, jakožto kombinací trojrozměrné informace prostředí a realistické barevné informace, lze očekávat zlepšení vnímání okolí robota operátorem.

Programová část práce vznikla pod záštitou projektu *Shadow Robotic System* na platformu *Care-O-bot*, o nichž je více napsáno v dalších sekcích 1.1 a 1.2. Navržené způsoby vizualizace scény jsou implementovány jako součást systému ROS (*Robot Operating System*), především prostředí Rviz, které je v tomto systému běžně používaným vizualizačním nástrojem.

V následující kapitole 2 budou přiblíženy problémy, které s sebou přináší ovládání robota na dálku a možnosti, jak se s těmito problémy lze vypořádat. V druhé sekci kapitoly 2 jsou popsány možnosti fúze dat z 3D senzoru a barevné kamery, a to od možností pořizování dat až po algoritmy k přímému texturování point cloudu. Následuje kapitola 3, zabývající se základy prjektivní geometrie, potřebnými pro praktickou část této práce.

V kapitole 4 je ve stručnosti popsán operační systém ROS využitý při tomto projektu. V jednotlivých částech kapitoly jsou popsány postupně konceptuální úrovně systému ROS, použitá prostředí pro simulaci a vizualizaci a použité technologie, relevantní pro tuto práci. Následuje oddíl, v němž jsou řešeny možnosti návrhu vizualizace a praktického řešení obou nabídnutých variant 5. Kapitola 6 popisuje podrobně všechny kroky, vedoucí k úspěšné implementaci variant vizualizace. V kapitole 7 jsou stručně shrnuty výsledky práce. V závěru je krátké zamyšlení ohledně využitelnosti práce, případně budoucích rozšíření 8.

1.1 Shadow Robotic System

Projekt *Shadow Robotic System* (dále SRS) je zaměřen na vývoj a výrobu prototypů dálkově ovládaných semi-autonomních robotických systémů pro domácí použití, především pro výpomoc starším lidem. Konkrétně SRS v současné době vyvíjí systém nazvaný „SRS robot“ pro osobní domácí péči [15].

SRS robot je navržen tak, aby byl schopen fungovat jako „stín“ svého operátora (odtud název projektu *Shadow Robotic System*). Je tím myšleno, že starší lidé mohou mít robota k dispozici jako stín svých dětí nebo pečovatelů. V tomto případě mohou operátoři robotů (tedy zmínění potomci nebo pečovatelé) pomáhat uživatelům na dálku a přesto fyzicky. Tento cíl, který si SRS vytyčil, by měl být realizován s pomocí následujících inovací:

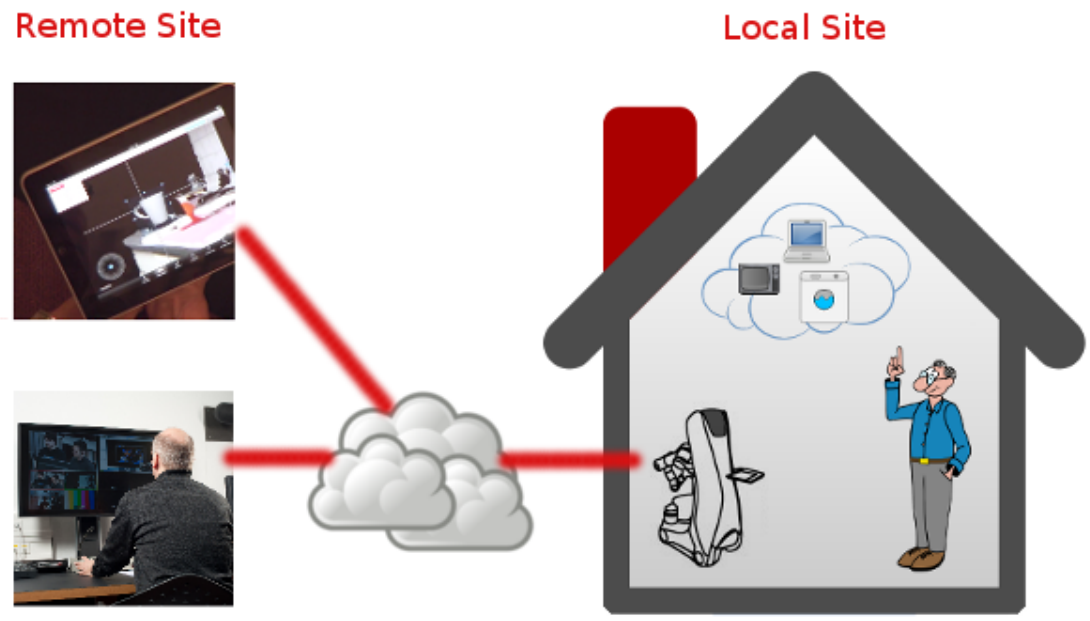
- Nový mechanismus vzdáleného ovládání, umožňující robotovi být ovládán přes existující komunikační síť.
- Adaptivní mechanismus řízení autonomity, umožňující efektivní plnění úkolů.
- Nový mechanismus samostatného učení robota, díky němuž může robot přizpůsobovat své chování podle svých předchozích zkušeností.
- Využití frameworku zaměřeného na bezpečnost, vzniklého s využitím rozsáhlých studií.

Systém ovládání robota je rozložen do třech vrstev. To znamená, že robota je možné kontrolovat přes tři různá uživatelská rozhraní. Na nejvyšší úrovni je možné robota ovládat příkazy zadávanými přímo na dotykovém panelu robota. Druhá úroveň uživatelských rozhraní bude provozována na zařízeních typu smartphone nebo tablet. Stále by mělo jít o vysokoúrovňové pokyny, přenášené robotovi bezdrátově. Na nejnižší úrovni by měl být robot ovladatelný vzdáleným specializovaným operátorem, který bude mít k dispozici veškeré senzorické informace a ovladací prvky robota. Čím nižší je úroveň ovladacího rozhraní, tím je třeba vyšší technická způsobilost operátora. Od koncových uživatelů se očekává použití nanejvýš prvních dvou vrstev rozhraní, nejnižší vrstva je určena vyškoleným pracovníkům. Princip vrstev uživatelských rozhraní je vyobrazen na snímku 1.1.

Výroba prototypů i celý projekt probíhá s podporou Evropské unie a podílí se na něm řada partnerů (viz internetové stránky projektu [10]).

1.2 Care-O-bot

Již více než deset let pracuje společnost Fraunhofer IPA ze Stuttgartu na vývoji mobilního robotického asistenta s označením *Care-O-bot*, schopného pomáhat lidem s jejich každodenními potřebami. V současnosti se pracuje na třetí generaci robotů označovaných *Care-O-bot*



Obrázek 1.1: Koncept vrstev ovládání systému Care-O-bot (převzato z [15])

3, která má veliký potenciál k tomu, aby mohla být úspěšně nasazena do každodenních prostředí.

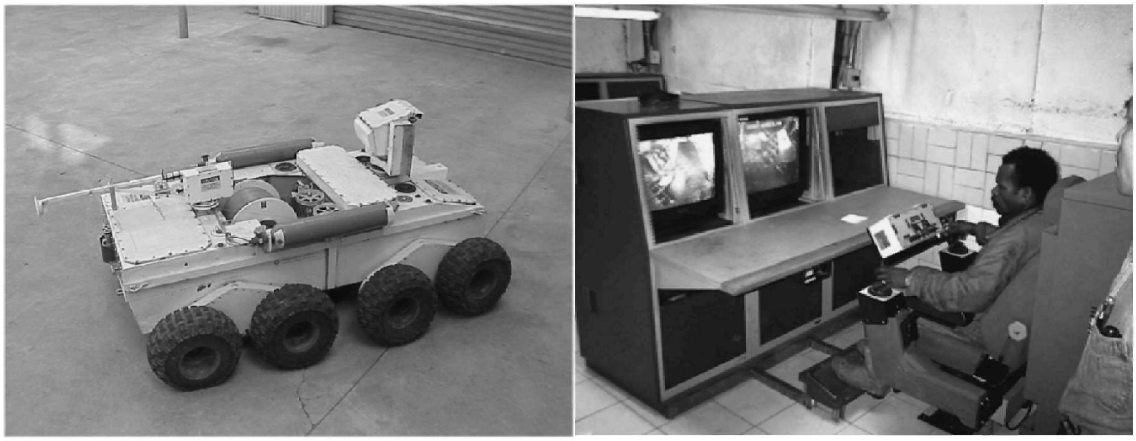
Jakožto interaktivní pomocník je Care-O-bot schopný bezpečně se pohybovat mezi lidmi, detekovat běžné předměty a také je uchopovat nebo předávat lidem. Care-O-bot může být také doplněn o aplikačně specifickou funkcionalitu například do firemních budov [10].

Platforma Care-O-bot 3 je 1.45 metrů vysoká na podvozku se čtyřmi koly. Manipulaci s předměty umožňuje jedno rameno s několika klouby a tříprstou „dlaní“ schopnou uchopovat objekty. Detekce okolí, překážek či předmětů je prováděna s pomocí množství senzorů, mezi nimiž nechybí barevná stereo kamera, laserový skener nebo 3D kamera. Transport objektů z místa na místo usnadňuje pohyblivý tác na přední straně těla Care-O-bota, s nímž může robot například servírovat jídla nebo nápoje. Při návrhu designu robota bylo záměrně upuštěno od existujících konceptů humanoidních robotických systémů a Care-O-bot by tedy neměl připomínat člověka [17].

Obrázek 1.2: Historie platformy Care-O-bot. Zleva Care-O-bot I, Care-O-bot II a Care-O-bot III (převzato z [10])

Kapitola 2

Vzdálené ovládání robota



Obrázek 2.1: Robot Numbat UGV firmy CSIRO pro těžební práce a jeho operátor (převzato z [18])

Vzdálené nasazení robotů přináší celou řadu možností využití samostatnosti robotů ve spolupráci s lidským operátorem. V současné době je vzdálené ovládání robota používáno pouze jako ovládání pasivního nástroje. Takovéto omezené využití potenciálu robotů s sebou přináší kromě horších výsledků také veliké nároky na pracovní schopnosti operátora. Zároveň je operátor obvykle vystavován veliké zátěži z důvodu omezených komunikačních možností s robotem, omezenou představou o okolí robota apod. [5].

2.1 Problémy spojené se vzdáleným ovládáním robotů

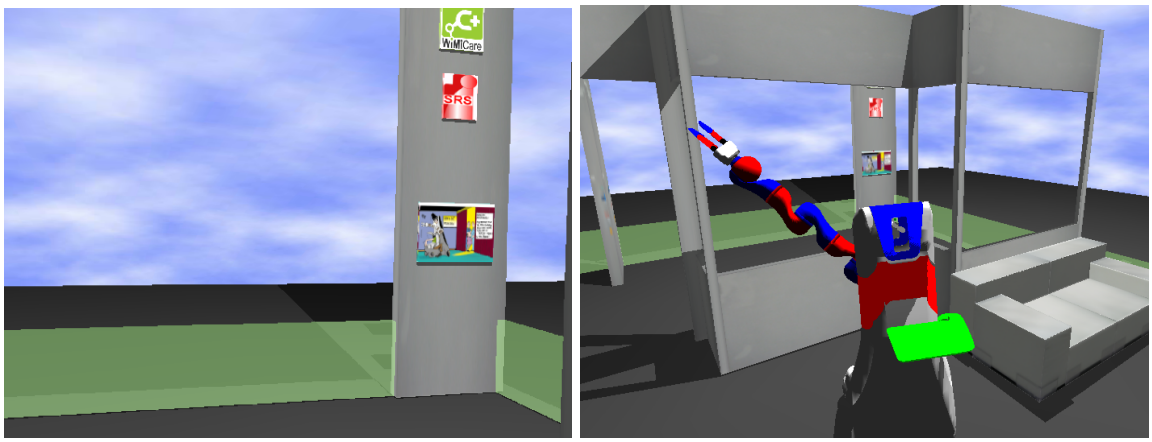
Současným trendem vzdáleného ovládání robotů je zkoumání možností společné kooperace robota a vzdáleného operátora na řešení zadaného úkolu - tzv. "Human-robot interaction"(HRI) [4]. Práce s robotem v zásadě spadá do dvou kategorií, vzdálené vnímání a vzdálené ovládání. Potíže s takovouto spoluprácí způsobuje několik faktorů. Především jsou výkony teleoperátora ovlivněny omezenými motorickými schopnostmi, schopností zůstat ve střehu a v neposlední řadě schopností vytvořit si mentální model prostředí, v němž robot existuje (odhady vzdáleností, detekce překážek apod.) [8].

2.1.1 Faktory ovlivňující vnímání okolí robota

V literatuře (Chen a další [8]) jsou shrnuty faktory, které nejvíce ovlivňují vzdálené vnímání okolí robota, do následujících bodů:

- Omezený zorný úhel

  Sledování prostředí pouze pomocí kamer přináš tzv. efekt "klíčové dírky"(keyhole effect). Znamená to, že operátorovi je představena pouze část reality zachycená kamerami, okolí operátor dostatečně nevnímá, nebo je třeba je dodatečně prozkoumat pohybem kamer. Tento omezený rozsah pohledu způsobuje na příklad blokování robotů o překážky, které nejsou viditelné ze současného úhlu kamery. Tato situace je vyobrazená na obrázcích 2.2. Na snímcích je Care-O-bot 3 stojící před překážkou, bránící mu jet vpřed. Na záběrech kamery však překážka není viditelná, protože se nachází níže, než je viditelný úhel kamery ze současné pozice. Tato situace může snadno vyústit v kolizi s překážkou, pokud nebude ovladací rozhraní robota doplněno o další prostředky vizualizace scény.



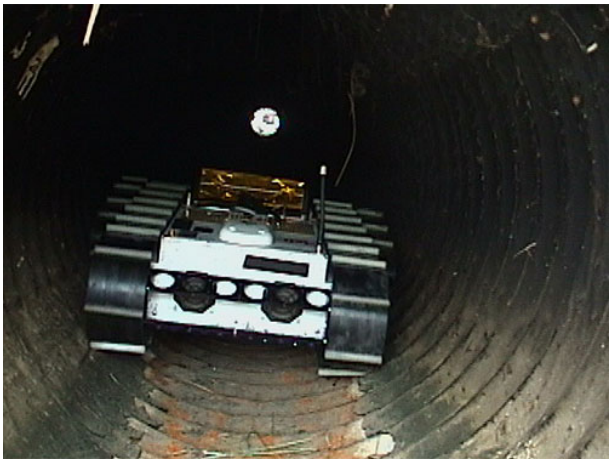
Obrázek 2.2: Znázornění omezeného zorného úhlu robotických senzorů. Aktuální snímek barevné kamery robota (vlevo) a pohled na scénu s robotem z větší vzdálenosti (vpravo)

- Poloha a orientace robota

  Časté potíže jsou způsobeny tím, že si operátor ze záběrů kamer dostatečně neuvědomuje orientaci robota v prostoru, především náklon vzhledem k některé z os robota (tzv. pitch a roll), což může vyústit v převrácení robota. Tento jev je dobře viditelný na obrázku 2.3, zobrazujícím záchranného robota. Z pouhého záběru kamery tohoto robota v této situaci nebude možné jednoznačně určit, jak je tělo robota nakolněno vzhledem k zemi. Tuto informaci lze odvodit na příklad z horizontu na záběrech kamery, nebo pozorováním objektů, na které zřetelně působí gravitace určitým směrem. Právě při záchranných pracech v troskách však tuto možnost operátor často nemá. K řešení tohoto problému se nabízí použití gyroskopů a akcelerometrů.

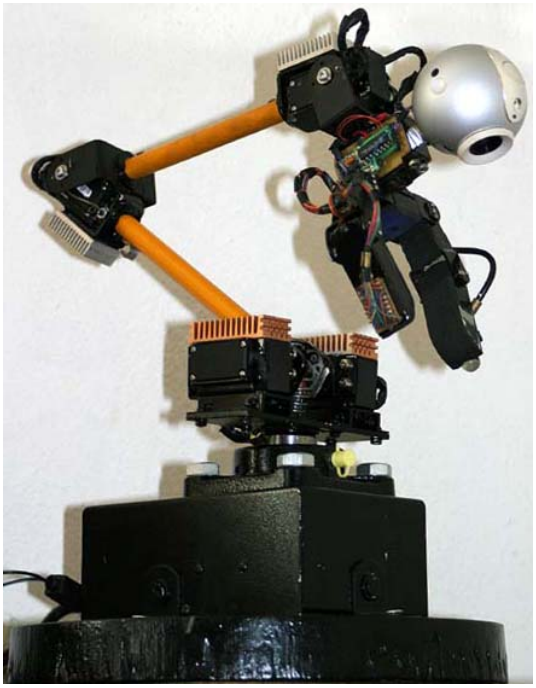
- Umístění kamery

  Kamery jsou nejčastějším prostředkem pro zobrazování prostředí robota. Mohou být umístěny jak na těle robota, tak na jednotlivých manipulačních prostředcích (ramenech, úchopech) či jiných částech robotů. Na obrázku 2.4 je robotické rameno s kamerou



Obrázek 2.3: Ukázka nejednoznačné orientace robota v prostoru (převzato z [7])

umístěnou těsně za uchopujícím zařízením. Takovéto pro člověka nepřirozené umístění zdroje obrazu může způsobovat dezorientaci. Často má operátor k dispozici také více obrazů s výsledky z různých senzorů (několika kamer, kinect, lidar atd.). Tyto rozšířené možnosti obecně zlepšují možnosti vnímání reality robota, problémem zůstává, že pozornost operátora může být v jeden moment upřena pouze na jediný obraz.



Obrázek 2.4: Robotické rameno s kamerou na úchopujícím zařízení (převzato z [16])

- Omezené vnímání hloubky

  V situaci, kdy jsou k dispozici jen záběry z monokulárních 2D kamer, je časté zkreslené vnímání hloubky prostoru. Konkrétně se toto zkreslení projevuje jako zkracování vnímaných vzdáleností. Tuto skutečnost demostrují obrázky 2.5. Na obou obrázcích je

fotografie téže chodby, ale na obou se může subjektivně jevit jinak dlouhá změnou úhlu pořízeného snímku. Způsob, jakým lze vyhodnotit správně vzdálenosti ze zmíněných snímků je na příklad porovnání délky chodby s objekty známých rozměrů, v tomto případě na příklad dveřmi.

Obrázek 2.5: Demonstrace problému s vnímáním hloubky záběrů z monokulární kamery

- Kvalita obrazu

  Experimentálně bylo zjištěno, že frekvence snímků z kamery, které má operátor k dispozici, nesmí být nižší než 10 Hz v případě, že je třeba vnímat nezkresleně pohyb robota v prostoru. Je-li třeba robotem provádět operace vyžadující manipulaci s objekty, je třeba frekvence alespoň 17,5 Hz.

- Zpoždení obrazu

  Podle experimentů ohledně vlivu zpoždění obrazu na schopnost provádět s robotem rozličné úkony bylo zjištěno, že negativní vliv na výkonnost operátorů má většinou zpoždění větší než 170 ms. Pochopitelně se výsledky liší podle zaměření zkoumaných úkolů (řízení, sledování okolí, manipulace s objekty atd.)

- Ovládání z pohybujícího se objektu

  Řadu nepříjemností přináší nutnost ovládat robota z pohybujícího se dopravního prostředku. Takováto nezvyklá kombinace vjemů se může projevit různě od snížení přesnosti ovládání robota, přes zpomalení reakcí až po nevolnost.

2.1.2 Možná řešení problémů s vnímáním okolí robota

Ve studii Chena a dalších [8] je nabídnuto řešení řady výše zmíněných problémů v podobě tzv. multimodálních displejů či multimodálního ovládání. Dnes je standardem okolí robotů prezentovat vizuálně a příkazy zadávat manuálně (tedy pomocí joysticků či klávesnice). Použití multimodálního zobrazování by umožňovalo zlepšit operátorům vznímání reality robota pomocí kombinace vizuální reprezentace v kombinaci s akustickými signály nebo dokonce haptickými (tedy dotykovými, jako na příklad vibrace pomocí haptických displejů). Stejně tak již dnes není nereálné ovládat roboty hlasovými příkazy či gesty pomocí optických nebo haptických zařízení, což může v budoucnosti přinést zlepšení v možnostech vzdáleného ovládání robotů.

Exsitují čtyři základní typy modelů ovládání robota z hlediska jeho samostatnosti [6]:

1. *Tele mode* je plně manuální mód, při němž operátor ovládá všechny akce robota. Robot je pouze pasivním nástrojem.

2. *Safe mode* je podobný Tele módu s tím rozdílem, že robot může v některých kritických okamžicích převzít iniciativu řízení, na příklad aby zabránil kolizi s překážkou.

3. *Shared mode* spočívá v dynamicky se měnících rolích mezi operátorem a robotem. Jde o mód, kterého se snaží návrháři robotů a ovladacích rozhraní dosáhnout k optimální kooperaci mezi robotem a operátorem. Robot může na příklad samostatně vyhledávat trasu na zadanou pozici a na rozcestích, nebo se u překážek dotazovat na instrukce operátora.

4. *Autonomous mode* je založen na definovaných vysokoúrovňových úkolech jako sledování trasy, hlídkování, prohledávání. Tyto úkoly je robot schopen plnit zcela samostatně a jediná zodpovědnost operátora spočívá v zadávání těchto úkolů.

Podle experimentů [4], kdy náhodně vybraní účastníci z řad žen i mužů a všech věkových kategorií, bez zkušeností s ovládáním robota, měli za úkol během šedesáti sekund prohledat uměle vytvořené prostředí pro pět vložených objektů, bylo dosaženo o poznání lepších výsledků při použití *Shared módu*, než při použití *Safe módu*. Vzhledem k tomu, že rozdíl výsledků nebyl patrný mezi muži a ženami ani mezi jednotlivými věkovými kategoriemi, je výsledek takovéhoto experimentu motivací zaměřit se na spolupráci robotů a jejich samostatnosti se vzdáleným operátorem.

Jiný experiment [4] měl za úkol objasnit, zda poskytuje operátorovi lepší představu o okolí robota záběr z kamery nebo virtuální 3D mapa. Polovině náhodných účastníků bylo dáno k dispozici rozhraní na obrázku 2.6. U druhé poloviny účastníků byl obraz kamery nahrazen virtuální 3D mapou 2.7. Oproti předpokladům nebyl ve výsledcích pozorovatelný znatelný rozdíl mezi použitím kamery a 3D mapy. Nicméně bylo zjištěno, že při použití 3D mapy byly úkoly dokončeny s průměrně výrazně menším množstvím pohybu ovladacího prvku a zároveň s menším množstvím chyb v navigaci. Tento experiment tedy ukázal, že lze s výhodou při vzdáleném ovládání robota použít 3D mapu okolí místo záběrů z kamery.

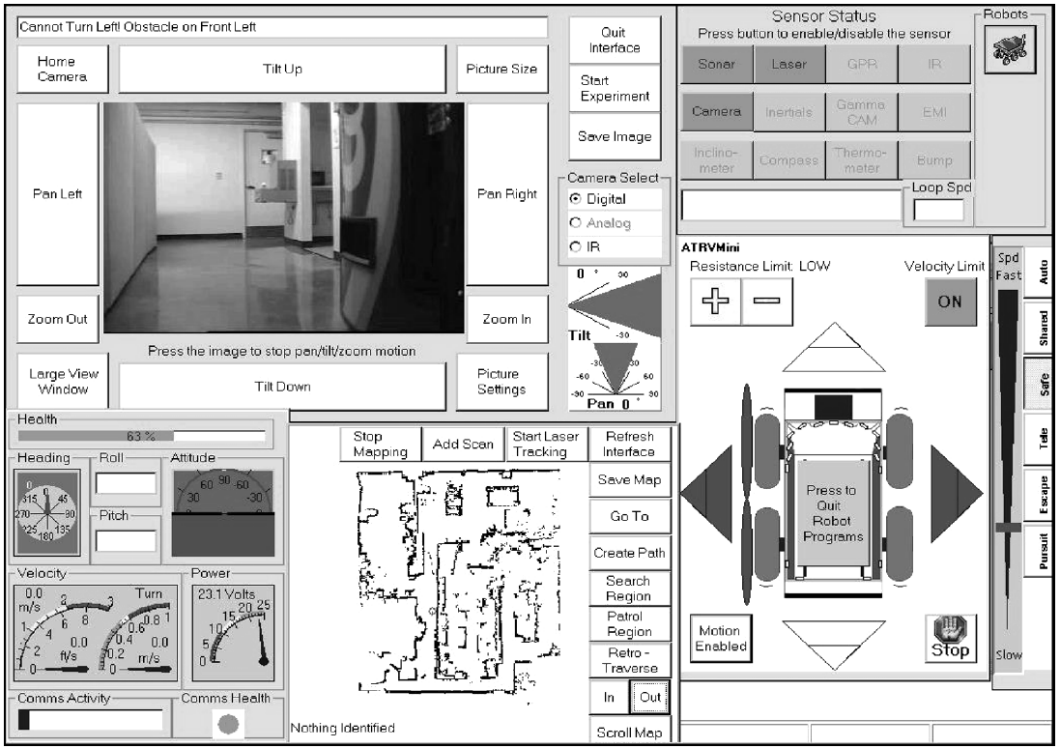
2.2 Fúze videa a point cloudu

Dvěma ze základních senzorů, poskytujících informaci o okolí robota, jsou 2D kamera a laserový skener. Dále budou popsány možnosti sjednocení výstupu těchto senzorů do jediného obrazu.

2.2.1 Techniky pořizování dat

Kamera je představitel tzv. pasivní technologie, zachycující již existující světlo ve světě, což ji činí závislou na momentálním osvětlení cíle, stínech apod. Oproti tomu laserový skener, jakožto aktivní technologie, vysílá vlastní laserový paprsek ke skenování světa. Tím je výsledek skenu nezávislý na momentálních podmínkách při skenování. Bohužel laserové skeny umí zjistit pouze pro každý bod obrazu jeho hloubku a hodnotu intenzity závislou na vlastnostech odrážejícího povrchu. Pro zjištění barvy musí být laser doplněn o jiný senzor, nejčastěji tedy o digitální kameru [1].

- Pevně připevněná kamera ke skeneru



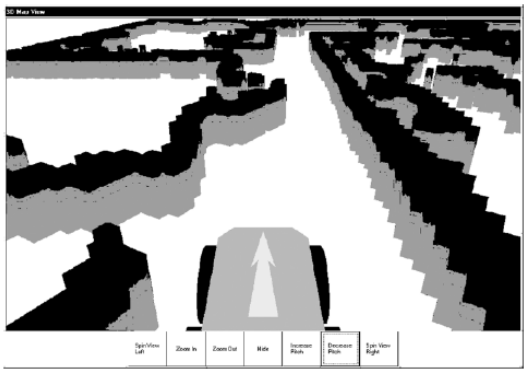
Obrázek 2.6: Rozhraní ovládání robota použité při experimentech popsaných v [4] (Převzato z [4])

První možností, která se nabízí, je připevnit digitální kameru přímo k laserovému skeneru. V tomto případě je mezi laserovým skenem a videem z kamery konstantní rozdíl zarovnání a korekce („registrace“ videa kamery a 3D skenu) může být provedena jedním výpočtem pro všechna získaná data. S takto získanými záběry je možné vytvořit model prostředí v podobě obarveného point cloudu. Tato medota je komplikována třemi základními problémy:

- *Závislost kamery na osvětlení* způsobuje, že při špatných světelných podmínkách je kvalita videa z kamery neadekvátní kvalitě skenu, který na osvětlení závislý není
- *Úhlový rozsah kamery*, který se pohybuje okolo 60°, omezuje skener, jehož rozsah může být až 300°
- *Dvoufázový sken*, tedy nejprve skenování hloubky laserem a následně snímání barev kamerou, může způsobit nepsrávnou detekci barvy point cloudu v případě zachycení pohybujícího se objektu na jednom ze skenů.

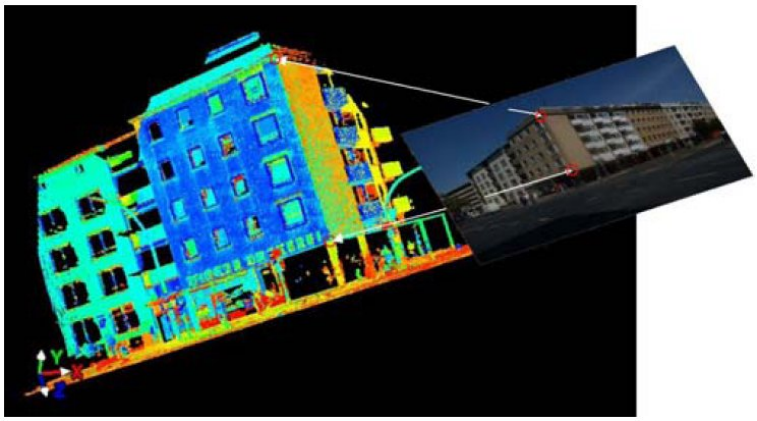
- Volná „ruční“ kamera

  Druhou variantou je nepřipevňovat kameru ke skeneru, ale obarvovat point cloud ze snímků pořízených ze samostatné kamery. Výhodou této metody je, že kamerové snímky nemusí být pořízeny ve stejnou chvíli jako sken. To umožňuje na příklad počkat na vhodné osvětlení, až zmizí pohybující se objekty a podobně. Pořizování kamerových záběrů nezávisle na skenu způsobuje nutnost registrace videa [21] do souřadného systému point cloudu použitím různých fotogrammetrických technik [1] [3].



Obrázek 2.7: Virtuální 3D displej (Převzato z [4])

Registrace snímků kamery a laserového skenu může být provedena různými způsoby. Nejčastější technikou je extrakce přirozených význačných bodů ve snímcích kamery, jako rohy budov, hrany a podobně. Stejné body by měly být nalezitelné i ve skenu a tedy je lze vzájemně namapovat. Tato metoda je viditelná na obrázku 2.8. Jinou možností je vložení umělých prvků do scény předtím, než jsou pořízeny záběry. Obvykle se jedná o bílé kruhy na černém pozadí. Tyto prvky jsou snadno rozeznatelné, což usnadňuje registraci skenu a fotografií. Toto řešení zjednodušuje automatizaci registrace, ale zásadně snižuje flexibilitu celého procesu získávání dat [2].



Obrázek 2.8: Registrace snímku kamery a point cloudu (Převzato z [1])

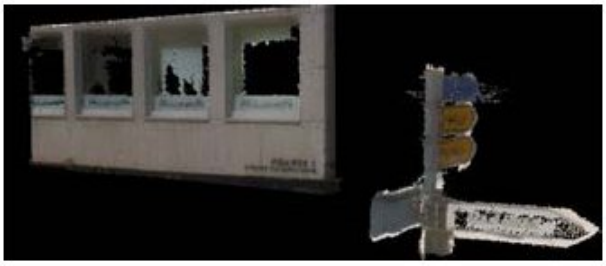
2.2.2 Obarvování point cloudu - Point Cloud Painter algoritmus

V momentě, kdy jsou k dispozici barevné snímky správně registrované na point cloud, je možná fůze těchto dat ze dvou různých senzorů [19]. Nejčastější je obarvování point cloudu barvama ze snímků kamery, čemuž nebrání nic jiného, než vzájemně zastíněné objekty. Problémem je tedy fakt, že si objekty na některých záběrech vzájemně stíní. V některých případech objekt některou částí stíní sám sebe (*self occlusion*). Takovéto zastiňování způsobuje při obarvování skenu jediným snímkem nesprávný výběr barvy tak, jak je vidět na obrázku 2.9 a 2.10. Takovéto nesprávné obarvování se obvykle řeší ruční korekcí a ručním

výběrem snímků z různých úhlů pro různé oblasti point cloudu. Opravování těchto chyb je viditelné na snímcích 2.10 a 2.11.

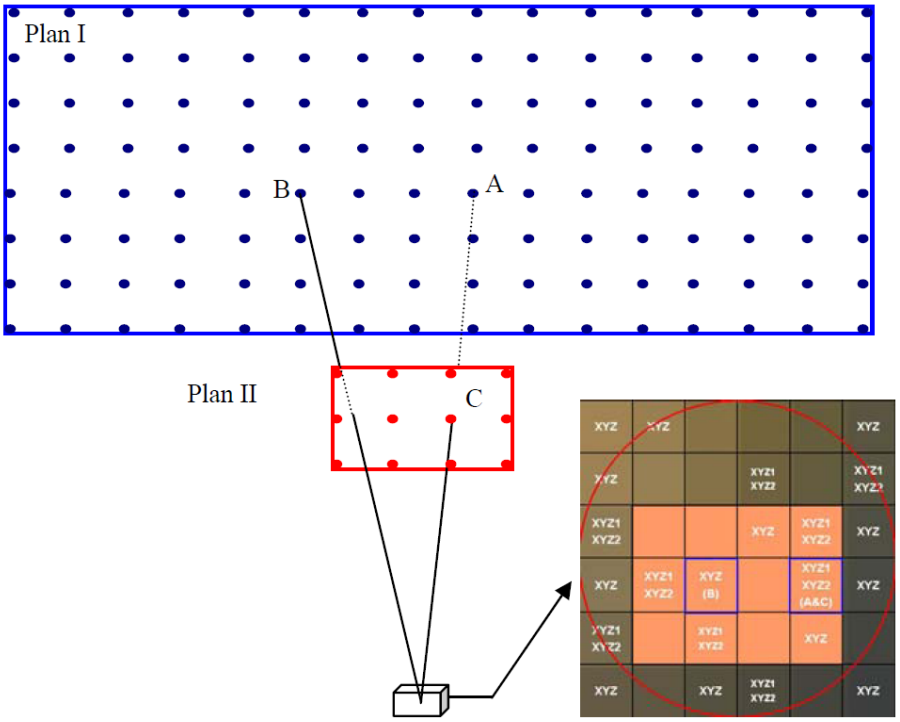
Obrázek 2.9: Point cloud a příslušný barevný snímek (Převzato z [1])

Obrázek 2.10: Obarvený point cloud s chybou a opravný barevný snímek (Převzato z [1])



Obrázek 2.11: Korektně obarvený point cloud (Převzato z [1])

Kromě ruční korekce chyb obarvování point cloudů je možné vyhodnocovat barvu jednotlivých bodů s pomocí série snímků scény z různých úhlů, automaticky. Algoritmus ošetřující problém stínění objektů a nesprávného obarvování se nazývá *Point Cloud Painter Algorithm* (PCP) [1]. Tento postup vychází z analýzy obrázku 2.12. Na obrázku jsou znázorněny dvě plochy zachycené na 3D skenu, označené *Plan I* a *Plan II*, které jsou ve vzájemném zákrytu (z pohledu kamery *Plan II* zakrývá část *Plan I*). V pravém spodním rohu obrázku 2.12 je znázorněn snímek kamery, jímž je point cloud obarvován. Všechny případy, které mohou nastat, demonstrují tři zkoumané body. Ve snaze obarvit snímkem bod C nedojde k ničemu neočekávanému, bod C přísluší rovině *Plan II*, která je v tomto místě vyobrazena i na snímku kamery, proto bude bod obarven korektně. Bod A na rovině *Plan I* se promítne na snímku do stejného pixelu, jako bod C. Přestože by se bod A měl obarvit nesprávně barvou roviny *Plan II*, lze tuto chybu detekovat vzdáleností těchto bodů od místa pohledu fotografie. Paprsek z bodu B neprotne *Plan II* v žádném konkrétním bodě, proto nebude zastínění bodu B rovinou detekováno a tento bod se nesprávně obarví barvou roviny *Plan II*.



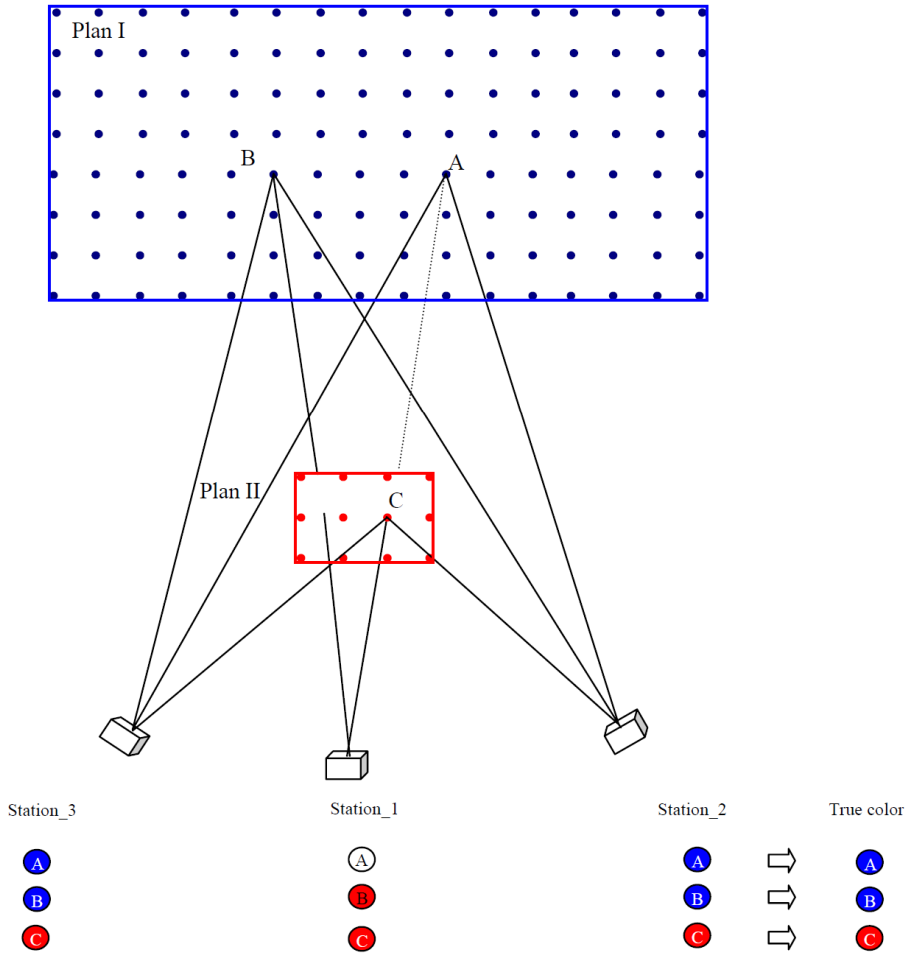
Obrázek 2.12: Nesprávné obarvení point cloudu v případě zastínění objektů (Převzato z [1])

Point Cloud Painter algoritmus bere point cloud ze skeneru a několik záběrů (snímků) barevné kamery, zachycujících stejný objekt jako sken, z různých úhlů. Princip vyhodnocení výsledné barvy zkoumeného bodu point cloudu znázorňuje obrázek 2.13. V něm je stejná scéna jako na obrázku 2.12, v tomto případě je však point cloud obarvován snímky ze třech různých stanic - *Station_ 1*, *Station_ 2* a *Station_ 3*. Značkami pod diagramem je vyjádřeno vyhodnocování barvy pro jednotlivé zkoumané body. V případě bodu C je situace jednoznačná, protože všechny 3 stanice zaznamenají pro tento bod stejnou barvu. Bod C je správně vyhodnocen jako červený. Bod A je vyhodnocen podle stanic *Station_ 3* a *Station_ 2* jako modrý, ze stanice *Station_ 1* se jeví bod jako červený, ale je možné detekovat chybu (paprsek pro zkoumaný bod protne i bližší bod point cloudu). Bod A je tedy korektně obarven modře. V případě bodu B je situace obdobná. *Station_ 2* a *Station_ 3* detekují korektní barvu, *Station_ 1* chybně. Tedy bod je obarven převažující detekovanou barvou, v tomto případě opět korektně modře.

Z popisu algoritmu je zřejmé, že každý bod, který bude na většině barevných snímků, které jsou k dispozici, viditelný nezastíněný, bude obarven správně. Algoritmus si nemůže poradit s případy, kdy je bod na větším počtu záběrů zastíněn jinou barvou, než bude snímků, na nichž bude barva správná..

Problematickou otázkou při obarvování point cloudu je také vyhodnocování, zda je barva bodu na dvou snímcích skutečně stejnou barvou, pozměněnou rozdílem světelných podmínek v jednotlivých úhlech pohledu, nebo zda se jedná o zcela jinou barvu. Pro tento účel byla vymyšlena tři kritéria, která musí být splněna, aby byly dva pixely považovány za pixely stejné barvy [1].

$$\Delta R \leq Criteria \wedge \Delta G \leq Criteria \wedge \Delta B \leq Criteria \tag{2.1}$$



Obrázek 2.13: Princip PCP algoritmu (Převzato z [1])

$$(\Delta R \geq 0 \wedge \Delta G \geq 0 \wedge \Delta B \geq 0) \vee (\Delta R \leq 0 \wedge \Delta G \leq 0 \wedge \Delta B \leq 0) \tag{2.2}$$

$$\Delta max. - \Delta min. \leq 0.75 \times (Criteria) \tag{2.3}$$

Zde *Criteria* je parametr, podle něhož může algoritmus vyhodnocovat, zda jsou dvě barvy přijatelně podobné. Obvyklá hodnota je mezi 10 a 20, podle míry tolerance světelných podmínek. ΔR, ΔG, ΔB jsou rozdíly jednotlivých barevných složek mezi body snímků. $\Delta max.$ a $\Delta min.$ jsou největší, respektive nejmenší rozdíl těchto hodnot (ΔR, ΔG, ΔB).

Postup obarvování bodů nakonec probíhá následovně:

1. Není-li bod zobrazený na žádné fotografii, není možné vyhodnotit barvu a neobarvuje se.

2. Je-li bod vyobrazen pouze na jedné fotografii, je otázkou nastavení algoritmu, zda se touto barvou bod obarví, nebo se pro nedostatek informací o barvě nechá bod neobarvený.

3. Bod je vyobrazen na více fotografiích. Pro každý pár fotografií jsou vyhodnocena kritéria podle rovnic 2.1, 2.2 a 2.3, zda jde o stejný bod (stejnou barvu). Vybere se nejčastěji vyskytující barva a bod se obarví průměrem dvou nejbližších z těchto barev.

Někdy point cloud neposkytuje dostatek informací o celistvých hranicích objektů. Proto lze fotografie scény také využít k doplňování míst mezi jednotlivými body point cloudu. Prozkoumáním bodů fotografií, které nebyly použity k obarvení point cloudu, je možné získat doplňující informace o scéně. Koordináty těchto bodů ve scéně je možné odhadnout z bodů jim blízkých, které se v point cloudu vyskytují a takto body přidávat a model zpřesňovat. Stejnou technikou je možné doplnit tzv. „mrtvé oblasti“, tedy místa, která nejsou skenerem vůbec zachycena, přestože na fotografiích se vyskytují [2].

Kapitola 3

Základy projektivní geometrie

V této kapitole je popsán způsob získávání snímků z kamerových senzorů při simulaci. Tento princip je označován jako model dírkové kamery (*pinhole camera model*). Jde o velice jednoduchý a zároveň velice efektivní model kamery sestávající z uzavřeného kvádru a jediného otvoru bez jakýchkoliv čoček. V další části kapitoly jsou popsány parametry, které tento model mohou přesně definovat.

3.1 Model dírkové kamery

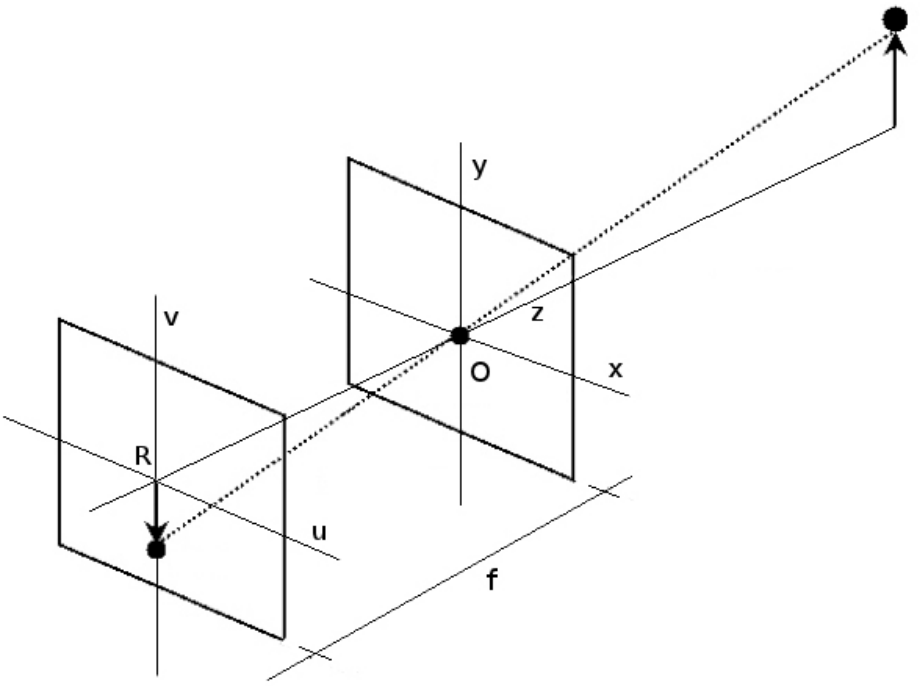
Dírková kamera je zatížená třemi limitacemi, vyplívajících z jejího samotného principu. První je omezené zorné pole. Další je fakt, že všechny paprsky musejí procházet otvorem kamery (*pinhole*). Třetí je uniformní vzorkování celé zobrazované oblasti, neumožňující vzorkovat flexibilně. Přes tyto problémy je dírková kamera nejčastěji používaným modelem při vizualizacích [20].

Model dírkové kamery určuje vztah mezi bodem v prostoru a bodem jemu příslušným v rovině obrazu. V případě dírkové kamery je toto mapování z 3D do 2D perspektivní projekcí. S modelem dírkové kamery je spojeno několik pojmů, které budou vysvětleny vzhledem k obrázku, znázorňujícím model kamery 3.1. Obrázek zázorňuje souřadný systém prostoru a jeho tři osy x, y a z se středem v bodě O, který je zároveň dírkou (*pinhole*) kamery. Od dírky je ve vzdálenosti f, která se nazývá *ohnisková vzdálenost* (*focal length*), umístěna rovina obrazu, určená osami u a v. Přímka, procházející dírkou kamery, která je kolmá na rovinu obrazu se nazývá *optická osa* (*optical axis*), která je v tomto případě shodná s osou z. Bod R, který je průsečíkem roviny obrazu s optickou osou bývá označován jako *hlavní bod* (*principal point*) [22].

3.2 Parametry kamery

Pro práci s takovýmto modelem kamery je třeba znát některé důležité rozměry. Pro výpočty parametrů potřebných pro tuto práci postačí znát výšku a šířku obrazové roviny, ohniskovou vzdálenost kamery a koordináty hlavního bodu. V prostředí ROS má každá simulovaná kamera k dispozici sadu parametrů, které obsahují jednak právě zmíněnou výšku a šířku obrazové roviny, model zkreslení a čtyři matice s doplňujícími parametry.

Matice `K` svým obsahem přesně odpovídá potřebným parametrům. Jednotlivá pole matice `K` znázorňuje rovnice 3.1, kde *fx* a *fy* jsou fokální (ohniskové) vzdálenosti v jednotlivých směrech. Jelikož je použit model planární dírkové kamery, jsou ohniskové vzdálenosti v obou



Obrázek 3.1: Model dírkové kamery

směrech stejné. Prvky cx a cy jsou souřadnice hlavního bodu kamery, tedy průsečíku optické osy s rovinou obrazu.

$$K = \begin{pmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Pro úplnost budou uvedeny i ostatní parametrické matice, přestože jich nebude v této práci dále využíváno.

Matice `D` obsahuje parametry zkreslení a obsah i rozměry této matice jsou závislé na použitém modelu zkreslení.

Rektifikační matice `R` se týká pouze stereo kamer. Jedná se o rotační matici, zarovnávající koordináty kamer do ideální roviny tak, aby epipolární linie byly pro oba stereo obrazy rovnoběžné.

Poslední maticí je projekční matice `P`. Její formát je uveden v rovnici 3.2. Jedná se o matici, která provádí projekci 3D bodu v souřadném systému kamery na 2D pixel v rovině obrazu kamery. U monokulárních kamer je obvykle $Tx = Ty = 0$ a $P[1:3, 1:3] = K$, tedy parametry uvedené v této matici jsou shodné s parametry matice `K`. Obecně se však tyto parametry mohou lišit. U stereo kamer souvisí poslední sloupec s pozicí optického centra druhé kamery vzhledem k souřadnému systému první kamery.

$$P = \begin{pmatrix} fx' & 0 & cx' & Tx \\ 0 & fy' & cy' & Ty \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.2}$$

S daným 3D bodem určeným souřadnicemi $[XYZ]'$ je projekce (x, y) tohoto bodu určena rovnicí 3.3.

$$\begin{aligned} \begin{bmatrix} u & v & w \end{bmatrix}' &= P \cdot \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}' \\ x &= u \ /w \\ y &= v \ /w \end{aligned} \tag{3.3}$$

Zde je matice $[uvw]'$ vyjádřením projekce bodu $[XYZ]$ v homogenních souřadnicích. Bod (x, y) je pouze jejím převodem do běžných dvourozměrných souřadnic.

Kapitola 4

Robot Operating System

Robot Operating System (ROS) je soubor nástrojů a knihoven k usnadnění tvorby robotických aplikací. Poskytuje prostředky k abstrakci hardwaru, ovladače, knihovny, prostředky k vizualizaci nebo komunikaci mezi procesy a podobně [12]. ROS je k dispozici pod BSD licencí, jde o open-source projekt. ROS se v jistých ohledech podobá jiným robotickým frameworkům jako Microsoft Robotics Studio [13] nebo Player [11]. Na rozdíl od obvyklých frameworků je cílem ROSu zaměřit se především na znovupoužitelnost kódu, což souvisí se strukturou aplikací (viz 4.1). ROS je vyvíjen na platformy na bázi Unixu, především Ubuntu.

4.1 ROS - koncepty

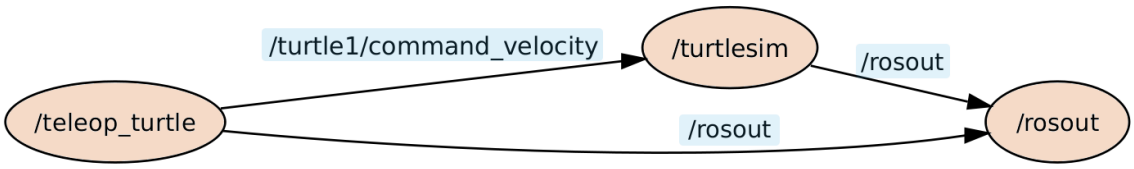
ROS má několik konceptuálních úrovní. Pro pochopení základních principů funkčnosti softwaru v ROSu je důležitá především úroveň souborového systému a výpočetní úroveň. Úroveň souborů se skládá z následujících prvků:

- *Package* je základní stavební jednotka veškerého software v ROSu. *Package* může obsahovat jak spustitelný proces (tzv. *node*), tak knihovnu, konfigurační soubor nebo jiná užitečná data.
- *Manifest* (`manifest.xml`) obsahuje metadata ohledně balíčku (*package*), informace o licenci, ale především závislosti na jiných balíčcích nebo přepínače kompileru apod.
- *Stack* je kolekce více společně souvisejících balíčků. ROSový software je obyčejně distribuován po *stack*ách.
- *Stack Manifest* je podobný jako obyčejný Manifest, ale týká se celého *stack*u (`stack.xml`).
- *Message type*, tedy typ zprávy, je definice struktury zpráv, jimiž balíčky komunikují.
- *Service type*, tedy typ služby, je definice komunikace s implementovanou službou. Definuje datovou strukturu pro požadavek (*request*) a odpověď na nej (*response*)

Výpočetní úroveň programů pro ROS (*Computation graph*) je peer-to-peer kolekce procesů, které zpracovávají data. Základními prvky jsou:

- *Node* jsou základní procesy, které provádějí výpočty. Většina programů pro robota se skládá z mnoha nodů, z nichž každý poskytuje nějaké výpočty (na příklad zpracovává data z jednoho senzoru, poskytuje plánování trasy a podobně.). Nody jsou implementovány za pomocí knihoven ROSu (příkladem pro jazyk C++ je to `roscpp`).
- *Master* je základní proces, který se stará o registraci jmen a obecně kontroluje komunikaci mezi ostatními nody (výměnu zpráv a podobně).
- *Parameter Server* umožňuje procesům centrálně ukládat data vyhledávatelná pod daným klíčem.
- *Message*, tedy zpráva, je základní komunikační prvek pro nody. Jde o datovou strukturu se zadanými prvky libovolných primitivních datových typů nebo polí primitivních datových typů, jejichž zasíláním mezi sebou si nody mohou předávat informace.
- *Topic* představuje prostředek pro zasílání a přijímání zpráv. Zasílání zprávy probíhá tím, že node publikuje zprávu (*publishing*) na nějaký konkrétní topic, čtení zpráv probíhá zaregistrováním daného topicu nodem pro příjem zpráv (*subscribing*). Proto je možné, aby zprávu publikovanou na topic četlo více nodů (všechny, které mají topic zaregistrovaný pro příjem), stejně tak do topicu může posílat zprávy více nodů.
- *Service* řeší potřebu komunikace typu požadavek/odpověď, pro kterou není systém message/topic ideální. Node implementující service ho poskytuje pod určitým jménem a klient využívající službu tak činí zasláním zprávy typu request.
- *Bag* je prostředek pro ukládání a opětovné přehrávání toku zpráv v ROSu. Jde o užitečný prvek obyčejně pro uložení zpráv ze senzorů robota a jejich opětovné vyvolání.

Komunikace mezi procesy je vyobrazena na obrázku 4.1, který je výstupem nástroje `rxgraph` a zobrazuje běžící procesy (Nody) a topicy, kterými spolu komunikují. Na obrázku jsou elipsami znázorněny nody, každá čára s šipkou je topic (začátek čáry je u nodu, který má topic zaregistrovaný pro *publishing*, konec čár je u nodu, který registruje topic pro *subscribing*). Nad každou čárou je název topicu, pod kterým jej nody registrují.



Obrázek 4.1: ROS - komunikace mezi procesy (Převzato z [12])

4.1.1 Vytvoření ROS package

Pro vytváření nových balíčků má ROS k dispozici konzolový nástroj roscreate-pkg s následující syntaxí:

```
roscreate-pkg <package-name> [dependencies...]
```

V seznamu *dependencies* je třeba vypsat všechny další balíčky ROSu, které bude tento balíček využívat, aby byly korektně připojeny, případně hierarchicky zkompilovány. Druhou možností je vložit nový node do již existujícího balíčku. V tom případě je třeba závislosti ručně dopsat do souboru `manifest.xml` mezi značky `<depend package="dependency"/>`, kde *dependency* je balíček, na kterém je nový node závislý. Mimoto je třeba informovat překladač o spustitelném souboru, který je třeba vytvořit. Tato informace se zapisuje do souboru `CMakeLists.txt` s danou syntaxí:

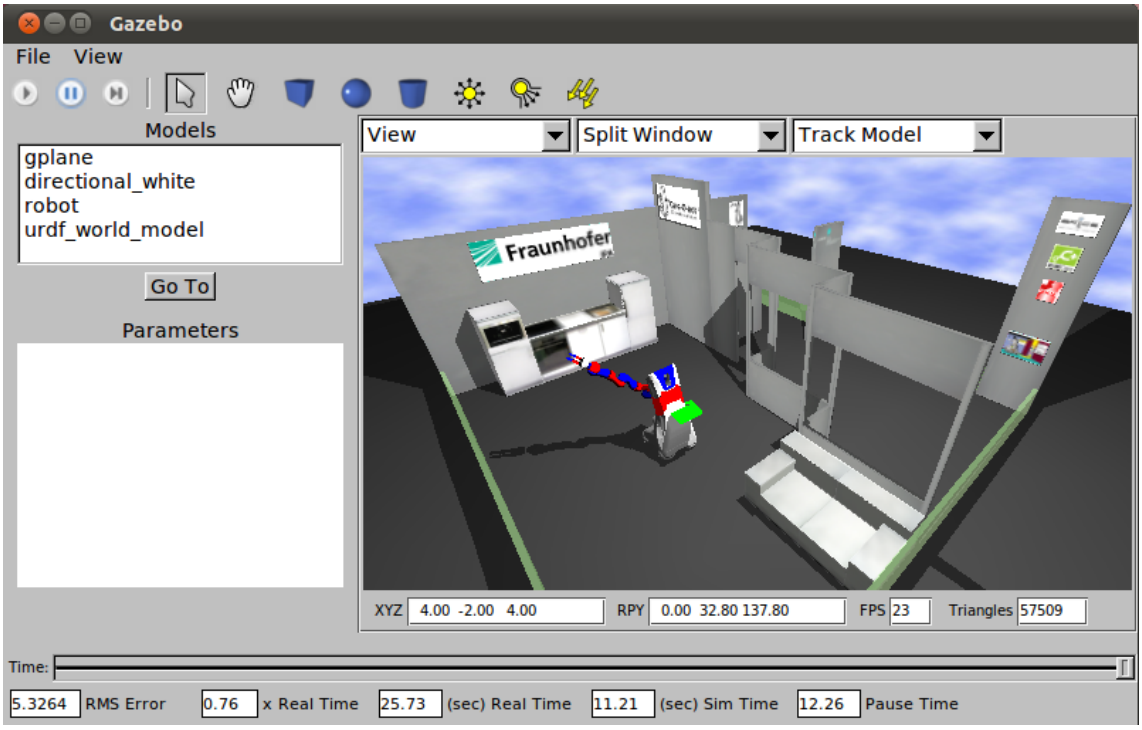
```
rosbuild_add_executable(název_nodu relativní_cesta_k_programu)
```

V neposlední řadě je třeba napsat program pro node a umístit jej na místo, kam se odkazuje přidaný řádek v `CMakeLists.txt`. Obvykle mají balíčky pro ROS pevně danou strukturu, tedy zdrojové soubory jsou umístěny ve složce `src/`, případné hlavičkové soubory ve složce `include/`.

4.2 Simulační a vizualizační prostředí

Tato část se zabývá popisem simulačního prostředí Gazebo a vizalizačního prostředí Rviz, používaných systémem ROS.

4.2.1 Gazebo



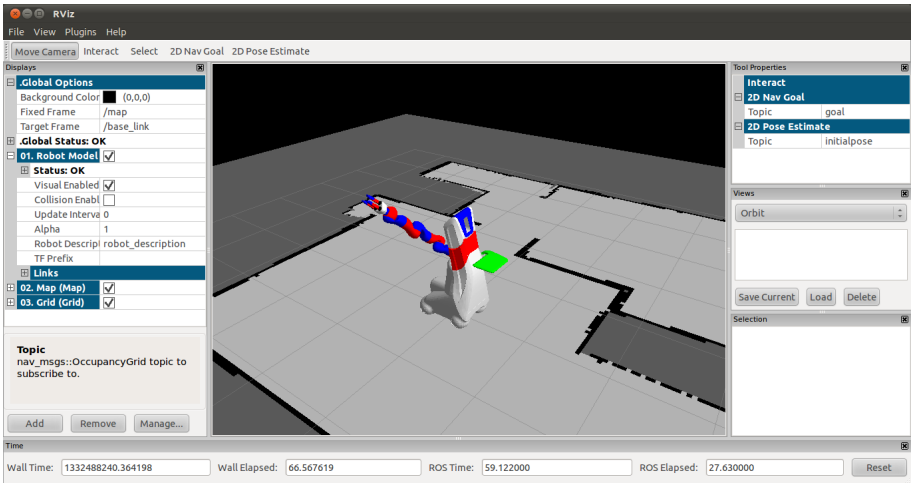
Obrázek 4.2: Okno simulátoru Gazebo se zapnutou simulací robota Care-O-bot 3

Gazebo je multi-robotický simulátor libovolných prostředí. Je schopno simulovat existenci robota a objektů ve třírozměrném světě. Dokáže také generovat realistickou odezvu senzorů a kolize objektů.

Gazebo se spouští obvykle s jedním povinným parametrem, kterým je `xml` soubor. Tento parametr je konfigurační soubor, který popisuje simulovaný svět a vše, co se v něm nachází (tedy včetně robota). Takovýmto spuštěním Gazebo zobrazí okno s pohledem na simulovaný svět s interaktivním ovládáním kamery pomocí myši [9].

4.2.2 Rviz

Rviz je prostředí pro vizualizaci robota s použitím ROSu. Na obrázku 4.3 je okno vizualizačního prostředí se zapnutou vizualizací robota. Dominantní okno uprostřed zobrazuje 3D pohled, na němž je momentálně vyobrazen robot na mapě a souřadné mřížce. Nalevo od něj je *Display list*, což je seznam načtených prvků, které se poté zobrazují ve 3D pohledu. Momentálně jsou v *Display listu* globální parametry zobrazení scény, model robota, mapa scény a souřadná mřížka.



Obrázek 4.3: Okno vizualizačního prostředí Rviz s vizualizací robota a mapy

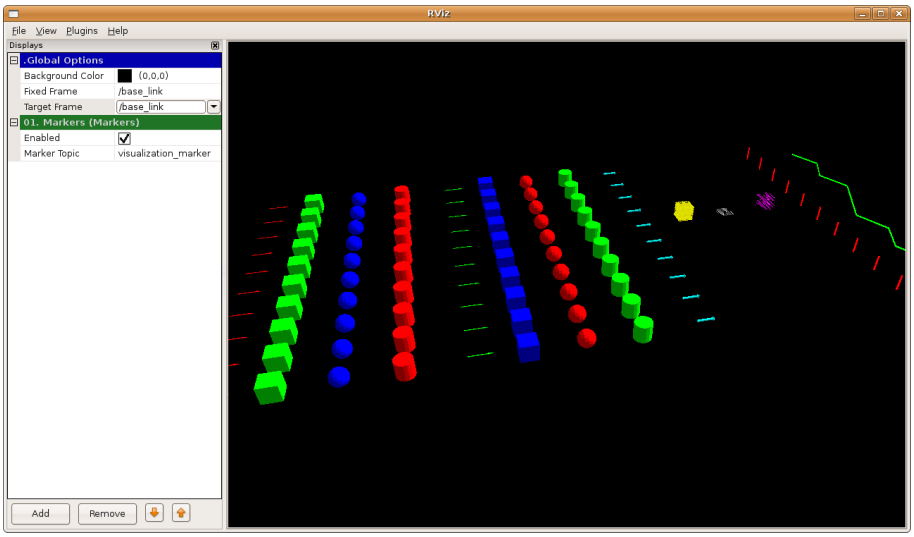
Přidáním prvků do *Display listu* se zaplňuje těmito prvky 3D pohled uprostřed okna. Přednastavené prvky (*Displeje*) v prostředí Rviz jsou na příklad *Camera* pro zobrazení videa (nevykresluje se do 3D pohledu, ale do nového renderovacího okna), *Laser Scan* zobrazuje data z laserového skeneru, *(Interactive) Markers* pro zobrazení (interaktivních) geometrických primitiv do 3D pohledu, nebo *Point Cloud* pro zobrazení point cloudu.

Každý prvek *Display listu* má sadu nastavitelných vlastností. Pro *display* typu Point Cloud jsou to na příklad možnosti zobrazení jednotlivých bodů, jejich velikost, barvy, ale především název *topicu* (viz 4.1), ze kterého má Rviz zprávy s point cloudem odchytávat. Na obrázku 4.3 jsou viditelné vlastnosti *displeje* „Robot Model“, jako obnovací interval, nebo průhlednost a podobně.

Mezi globálními nastaveními jsou důležité především dva parametry pro nastavení souřadnicových rámců (*frames*). *Fixed frame* je pro správné zobrazování důležitější. Jde o referenční rámec, určující, k čemu se vztahuje poloha okolního světa. Není striktně dáno, čím musí být *fixed frame* určen, pro správné zobrazení je však nutné, aby šlo o nějaký prvek světa, který se ve světě nepohybuje a má tedy fixní pozici. *Target frame* je referenční rámec pohledu kamery. Tedy pokud *target frame* je mapa, zobrazí se pohybující se robot ve statické mapě.

Pokud je *target frame* na příklad podvozek robota, bude zobrazen statický robot a svět se bude pohybovat relativně okolo něj.

Zajímavou možností vykreslování do 3D pohledu prostředí Rviz jsou tzv. *markery*. Markery umožňují umístit do 3D pohledu přednastavená geometrická primitiva zasíláním zpráv typu `msg/Marker` nebo `msg/MarkerArray`. *Display* typu marker má jednu nastavitelnou vlastnost, kterou je *topic*, na němž má Rviz očekávat zprávy s vlastnostmi markerů (obvykle se jedná o tvar, barvu, velikost, pozici). Použití markerů zobrazených v 3D pohledu v prostředí Rviz ukazuje obrázek 4.4.



Obrázek 4.4: Rviz - zobrazení geometrických primitiv pomocí Markerů (Převzato z [12])

4.3 Využívané technologie

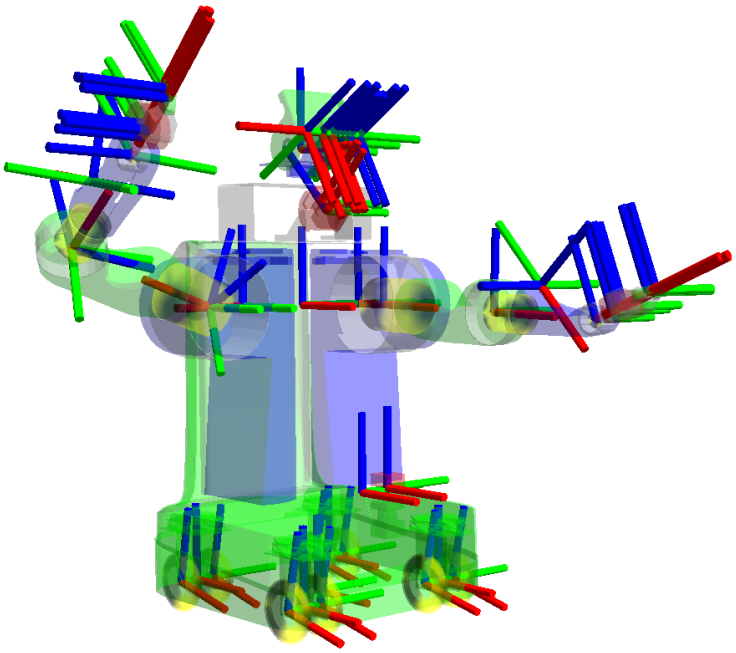
Následuje popis knihoven nezbytných pro funkčnost programů implementovaných v této práci.

4.3.1 Balíček TF

TF je balíček ROSu, vytvořený pro zjednodušení práce s více souřadnými systémy ve scéně, které se navíc mohou měnit v čase. TF si uchovává informaci o vztazích mezi jednotlivými souřadnými systémy a jejich časový průběh si ukládá do bufferu. Vytváří tak časový záznam o stromové struktuře souřadných systémů, čímž umožňuje uživatelům používat transformace bodů, vektorů a podobně, mezi libovolnými souřadnými systémy v požadovaném čase.

Robotický systém se typicky skládá z mnoha 3D souřadných systémů v tomto kontextu nazývané *rámce* (*frames*), které se mění v čase. Typicky se jedná o rámec scény, podvozku, konkrétního kloubu ramena, kamery a podobně. TF udržuje záznam o všech jejich vztazích a změnách. Díky tomu je možné zjišťovat za pomoci TF na příklad jaká je pozice podvozku robota ve scéně, jaká je vzdálenost ramena od objektu, který je třeba uchopit, nebo jaká byla poloha hlavy robota vzhledem k podvozku před pěti sekundama.

Přestože TF je primárně knihovna použitelná v kódech jednotlivých procesů ROSu (nodes), zpřístupňuje i několik konzolových nástrojů. `rosrun tf tf_monitor` vypíše do



Obrázek 4.5: Zobrazení několika souřadných systémů spojených s tělem robota (Převzato z [12])

konzole informace o celém aktuálním transformačním stromu. Je možné si vypsat informace o monitorování transformace mezi dvěma zadanými souřadnými systémy příkazem `rosrun tf tf_monitor <source_frame> <target_frame>`, případně je možné nahlédnout na konkrétní hodnotu transformace (tedy vektory posunu a rotace) mezi dvěma rámci - `rosrun tf tf_echo <source_frame> <target_frame>`.

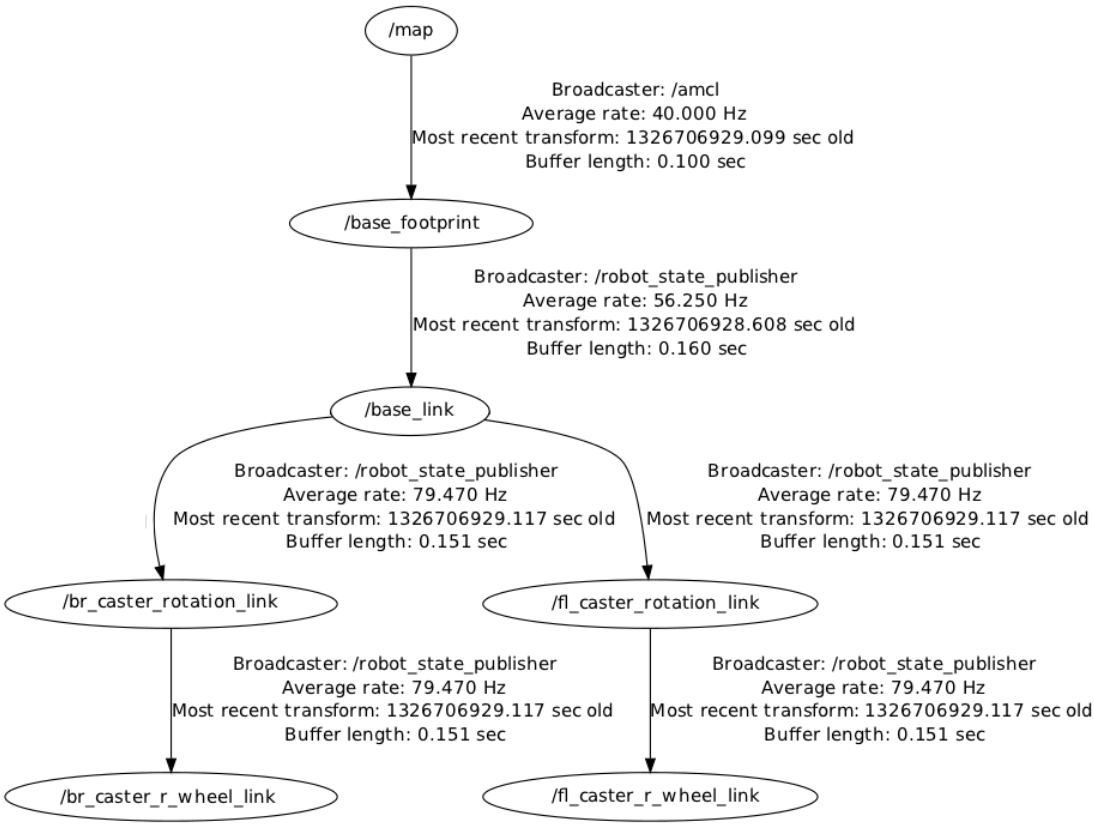
Nejnázornějším nástrojem v rámci balíčku TF je graf rámců, který je možné vytvořit příkazem `rosrun tf view_frames`. Po zavolání příkazu je k dispozici PDF soubor `frames.pdf`, obsahující vizualizaci orientovaného stromu transformací a vztahy mezi jednotlivými rámci jsou reprezentovány hranami v tomto grafu. Zjednodušený nákres takového stromu zobrazuje obrázek 4.6, který by reprezentoval systém s rámcem vztaženým ke scéně v kořeni stromu a několik dalších rámců spojených s podvozkem a jednotlivými koly podvozku.

4.3.2 OGRE

Vizualizační prostředí Rviz 4.2.2 používá k renderování vizualizace grafickou knihovnu Ogre. Aby bylo možné upravovat grafické prvky, které protředí Rviz vykresluje, je vhodné alespoň základním principům této knihovny rozumět.

Ogre je zkratka celého názvu *Object-Oriented Graphics Rendering Engine*, jde tedy o objektově orientovaný grafický renderovací engine. Jedná se o nástroj napsaný v C++ a slouží k usnadnění a více intuitivnímu vytváření aplikací s použitím hardwarově akcelerované grafiky. Knihovna Ogre je abstrakcí nad systémovými knihovnami jako DirectX nebo OpenGL, skrývá implementační detaily nižších vrstev a poskytuje rozhraní k intuitivnímu vytváření grafu scény pomocí existujících objektů [14].

Ogre není herní engine, poskytuje pouze grafické prostředky pro vytvoření a renderování scény. Neumožňuje obohatit aplikaci o prvky jako zvuk, síťové služby, umělou inteligenci,



Obrázek 4.6: Zjednodušený výstup nástroje view_frames se orientovaným stromem transformací

kontrolu kolizí nebo reálnou fyziku.

Diagram 4.7 ukazuje některé nejdůležitější objekty, vyskytující se v aplikacích využívajících Ogre. Na úplném vrchu diagramu je objekt *Root*. Ten funguje jako vstupní bod do systému Ogre a pomocí něj se vytvářejí objekty nejvyšších úrovní jako manažery scén, renderovací systémy, renderovací okna, pluginy a podobně. Obvykle objekt Root poskytuje objekty, které teprve udělají práci, která je požadována, Root sám o sobě je tedy spíš organizačním prvkem. Většina dalších objektů v Ogre spadá do jedné ze třech kategorií.

- Management scény

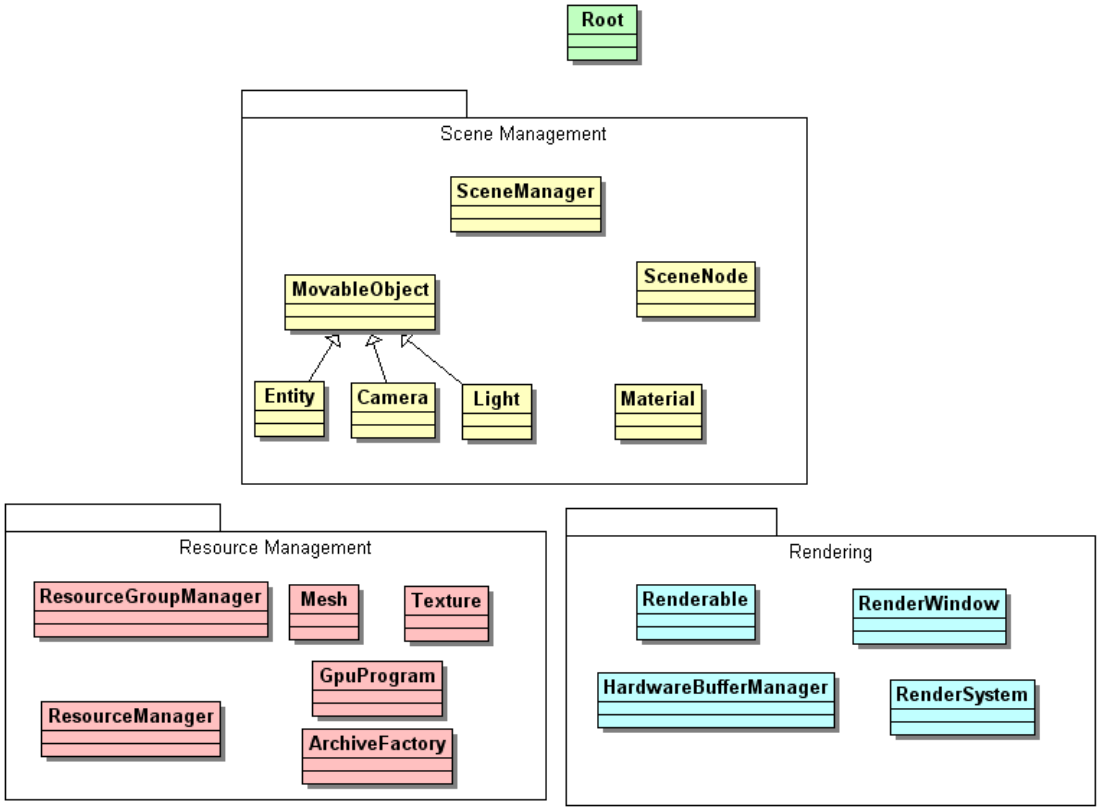
  Jde o jednotlivé reálné objekty ve scéně, struktura scény, pozice a orientace kamery a podobně. Objekty této kategorie poskytují přirozené rozhraní ke světu, který je modelován scénou.

- Management zdrojů

  Zdroji (*resources*) se v tomto kontextu rozumí data, která jsou nezbytná ke správnému vykreslení scény mimo samotnou strukturu rozmístění objektů, tedy geometrie objektů, textury, fonty a podobně. Objekty této kategorie se starají o to, aby byly tyto zdroje korektně načítány, efektivně využívány nebo odstraňovány z pracovní paměti.

- Renderování

  Objekty týkající se nižších vrstev renderovací pipeline. Nejčastěji se jedná o specifické systémové objekty jako buffery a podobně.



Obrázek 4.7: Diagram některých důležitých objektů v Ogre (převzato z [14])

Mimo tyto typy objektů může být systém Ogre obohacen také o objekty typu *plugin*. Ogre bylo navrženo, aby bylo snadno rozšiřitelné a právě pluginy jsou prostředky k těmto rozšířením. Z většiny tříd Ogre lze vytvářet podtřídy a tak je rozšiřovat. Může se jednat o nový *SceneManager* s vlastní organizací scény, novou implementaci renderovacího systému nebo třídu poskytování načítání zdrojů z nestandardního umístění (na příklad z webového umístění nebo databáze). Díky těmto možnostem se Ogre řadí mezi renderovací prostředky s velice univerzálním využitím, umožňující vykreslit prakticky jakoukoliv scénu [14].

Z pohledu této práce mají větší význam objekty skupiny Managementu scény a zdrojů. Kromě objektu Root, který by měl být prvním vytvořeným objektem v aplikaci Ogre, je zřejmě nejdůležitějším prvkem *SceneManager*. Ten se stará o organizaci veškerého obsahu scény, o vytváření a spravování kamer, pohyblivých objektů (entit), světel a materiálů (povrchových vlastností objektů, nikoliv textur). Není třeba si uchovávat žádné seznamy entit nebo kamer. Vše je k dispozici prostřednictvím SceneManageru a jeho metod jako `getLight`, `getCamera` a podobně. Je to také objekt SceneManager, který zasílá prvku *RenderSystem*, co má vykreslit. Obvykle však není třeba manuálně volat metodu `SceneManager::_renderScene`, která má tuto činnost na starost, ale renderovací cíl si ji zavolá sám vždy, když potřebuje obnovit obraz. Základní objekt SceneManager implementuje funkčnost popsanou výše, ale organizace

scény v něm je pouze velice jednoduchá. Je to způsobeno velikou rozmanitostí typů scén a požadavků na jejich spravování. Proto není možné očekávat dobré výkonostní výsledky základního SceneManageru na velikých scénách. Pro tyto účely je třeba použít některý ze specializovaných typů, na příklad `BspSceneManager`, který je optimalizovaný pro velké scény organizované do BSP stromu (*Binary space partition tree*).

Třída *ResourceGroupManager* je prostředek pro správu zdrojů jako textury nebo meshe. Tato správa probíhá prostřednictvím několika ResourceManagerů, které se zabývají vždy konkrétním typem zdroje, jako *TextureManager* nebo *MeshManager*. ResourceManagery zajišťují, že jsou zdroje načítány vždy jen jednou a sdílené v celém Ogre engine a zároveň spravují paměťové požadavky zdrojů.

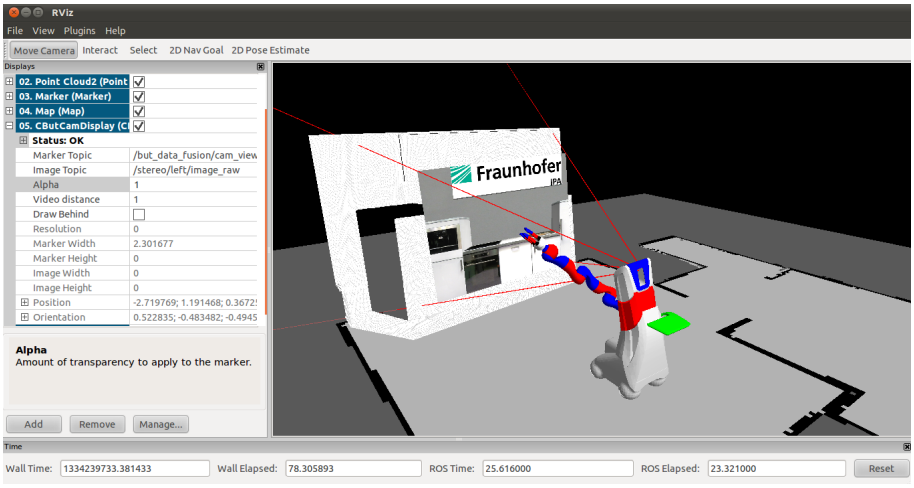
Kapitola 5

Návrh vizualizace pro ovládání robota

V počítačové grafice je poměrně obvyklou technikou fúze 3D point cloudu a barevných snímků pro získání barevného 3D modelu scény 2.2. Zároveň je běžnou praxí využívat k ovládání robotů jak barevný video stream, tak 3D mapu z laserových senzorů 2.1. Nabízí se tedy možnost využití k ovládání robota kombinaci šedotónové 3D mapy a barevného videa pro dosažení potenciálně lepší orientace vzdáleného operátora v okolí robota.

Z možností fúze point cloudu a barevného videa byly vybrány následující:

- *Zobrazení videa na poloprůhledném polygonu před point cloudem.*



Obrázek 5.1: Zobrazení videa na poloprůhledném polygonu před point cloudem

Za pomoci znalosti pozice robota a jeho orientace ve scéně je možné vykreslit do prostoru polygon s dynamickou texturou, znázorňující vždy aktuální snímek kamery. Při tomto návrhu řešení je třeba řešit následující otázky:

– Vzdálenost vykreslování polygonu od robota musí být dynamická.

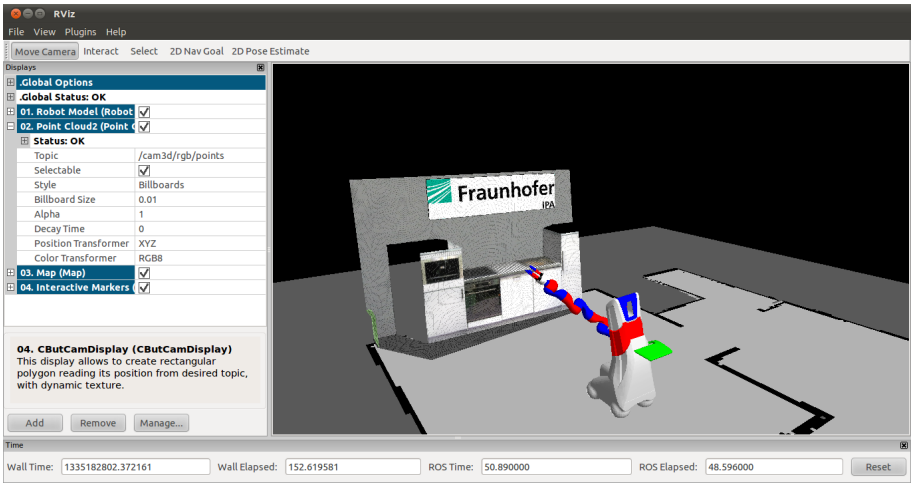
  S měnící se vzdáleností point cloudu od robota by se při pohybu robota při konstantní pozici polygonu vzhledem k robotovi mohl obraz videa dostat za point cloud, proto je třeba jeho pozici vykreslování měnit podle aktuální situace.

– Vykreslený polygon může zastínit velkou část point cloudu.

Tímto by při pohledu do scény nebyla viditelná znatelná část hloubkové informace o scéně. Řešením může být částečná průhlednost polygonu se záběrem videa. To zviditelní jak barevnou informaci, tak hloubkovou informaci, ale zobrazením dat takto přes sebe lze očekávat znepřehlednění vizualizace.

Více o tomto návrhu řešení v sekci 5.1

- *Obarvení jednotlivých bodů point cloudu.*



Obrázek 5.2: Obarvený point cloud

Jsou-li k dispozici dvě barevné kamery a 3D point cloud scény, je možné se pokusit jednotlivé body point cloudu obarvit, například s využitím Point Cloud Painter algoritmu 2.2.2 [1]. U této varianty lze předpokládat následující vlastnosti:

– Tímto sloučením informace o barvě i hloubce se zřejmě zpřehlední a zjednoduší vnímání těchto informací operátorem. Vizualizace je takto přirozenější a podobnější realitě.

– Rozlišení point cloudu je mnohem menší, než rozlišení barevných snímků. Při obarvování bude třeba řešit otázku zda obarvovat bod point cloudu jeho nejbližším pixelem, nebo barvu interpolovat z okolních pixelů. Vždy však dojde ke ztrátě části barevné informace.

Více o tomto návrhu řešení v sekci 5.2

5.1 Zobrazení videa na polygonu

První variantou, jak sjednotit point cloud a video do jediného renderovacího okna je možnost zobrazit video stream na polygonu před robotem. Představa řešení je taková, že před robotem bude zobrazené pohledové těleso (jehlan nebo komolý jehlan) a v tomto pohledovém tělese bude zobrazováno barevné video z kamery robota na obdélníkovém polygonu. Video by mělo měnit framy v reálném čase tak, jak přicházejí ze senzorů robota a celý polygon s videem i pohledové těleso bude měnit svou pozici a natočení tak, aby se vždy nacházelo před robotem a mířilo do směru jeho pohledu. Nákres této varianty řešení je znázorněn na obrázku 5.5, kde

je červeně vykresleno pohledové těleso a v něm zobrazen poloprůhledný polygon s aktuálním snímkekm barevné kamery. Snímky, z nichž nákres vychází, jsou na obrázku 5.4, kde je vizualizace simulovaných dat z 3D senzoru robota, a obrázku 5.3, kde je aktuální záběr jedné z kamer.

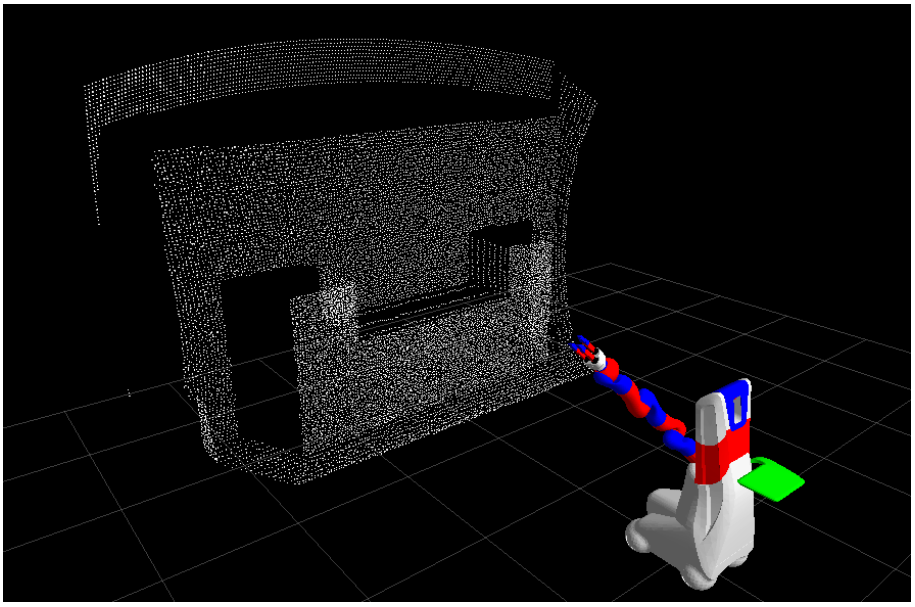
Obrázek 5.3: Jeden snímek pravé kamery robota

5.1.1 Možnosti řešení zobrazení videa na polygonu

Technické možnosti řešení nabízí vizualizační prostředí Rviz popsané dříve v kapitole 4.2.2. Zobrazení point cloudu je možné přímo ze senzorů přes přednastavený typ *displeje*. Pohledové těleso je možné zobrazit s využitím markerů 4.2.2 typu úsečka.

Komplikovanější situaci způsobuje zobrazení videa. Triviálním řešením je odchytávání framů videa a zobrazování jednotlivých pixelů jako markerů typu bod, obarvených barvou pixelů. Každý marker se však vytváří a mění pomocí zpráv, což by i při nízkém rozlišení videa znamenalo veliký komunikační tok mezi vizualizačním prostředím a procesem zpracovávajícím video. Lepším řešením se zdá zobrazovat video jako marker typu polygon s texturou v podobě aktuálního framu videa. Pro ušetření datového toku potřebného pro komunikaci by bylo ideální, kdyby byl marker sám schopný odchytávat framy s texturou z video topicu. Pro tyto účely by bylo třeba zderivovat implementaci markeru pro vlastní potřeby a použít ve vizualizačním prostředí Rviz jako plugin. Způsob vytvoření pluginu je popsán dále v kapitole 6.1.4.

Další otázkou při tomto řešení fúze je vzdálenost, v jaké se má video frame před robotem zobrazovat. Aby pasoval polygon s videem do pohledového tělesa, čím blíže bude robotovi, tím bude muset být menší. Proto se zdá být lepší zobrazovat video dále od robota, nicméně maximální vzdálenost zobrazení polygonu bude omezena okolím robota, tedy vzdáleností point cloudu před robotem. Tak by mohlo docházet k nežádoucím jevům, jako zobrazení



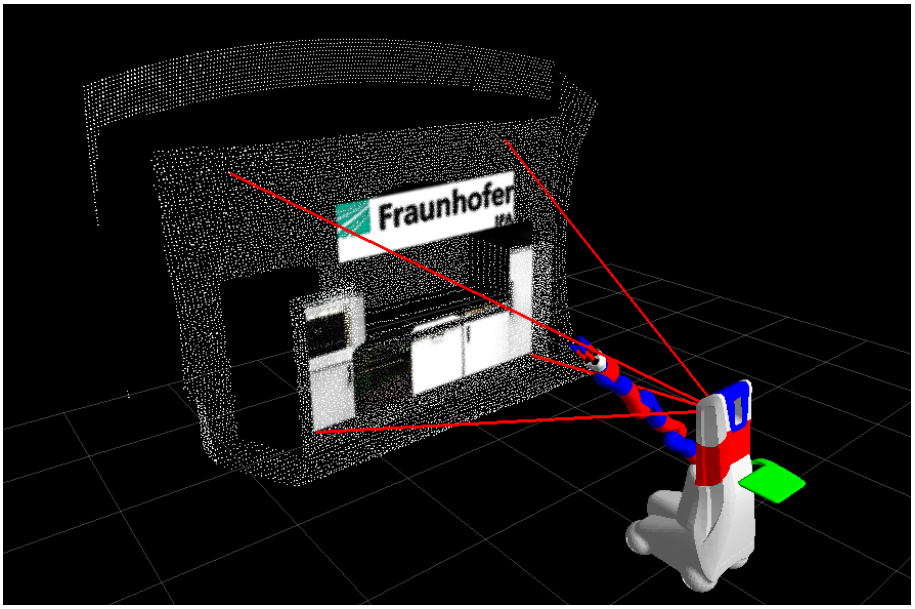
Obrázek 5.4: Vizualizace simulovaných dat z laserového skeneru robota

videa před robotem za stěnou nebo jiným objektem. Vzdálenost videa se tedy bude muset dynamicky měnit podle okolí robota. Toho bude možné dosáhnout relativně snadno průchodem jednotlivými body point cloudu a vyhledáním nejbližšího z nich, podle jehož hloubky bude odvozena maximální vzdálenost vykreslování polygonu.

5.1.2 Komponenty vizualizace

K tomu, aby mohly být prováděny výpočty na straně robota, je třeba vytvořit *ROS Package* a v něm napsat program pro *ROS Node* (viz ROS koncepty 4.1 a vytvoření ROS package 4.1.1).

V momentě, kdy je vytvořený spustitelný node a je jasné, které výpočty bude provádět, je možné se zamyslet nad předpokládaným datovým tokem mezi jednotlivými zůčastněnými prvky. Tato propojení znázorňuje obrázek 5.6. V něm jsou znázorněny již i zprávy potřebné pro dynamicky texturovaný polygon. Pro potřeby zobrazení pohledového objemu je důležitý prvek *view*, což je jméno procesu (*node*), který počítá veškeré potřebné parametry pro vizualizaci na straně robota. Šipky směřující do procesu jsou odebírané zprávy s potřebnými informacemi. *CameraInfo* jsou parametry kamery, z nichž lze spočítat parametry pohledového objemu. *Tf* umožňuje získat transformace mezi rozdílnými souřadnými systémy senzorových dat a je tedy potřeba ke správnému umístění pohledového objemu vzhledem ke scéně. *PointCloud2* jsou data z 3D senzoru, které je potřeba vzít v úvahu až při tvorbě texturovaného polygonu. Šipky směřující z procesu *view* jsou zprávy, které posílá vzdálenému operátorovi, konkrétně vizualizačnímu prstředí Rviz. *Marker* je standardní zpráva ROSu typu `visualization_msgs::Marker` pro zobrazení pohledového objemu. `ButCamMsg` je zpráva s informacemi o texturovaném polygonu potřebná pro zobrazení displeje `But_cam_display` a bude vysvětlena později. Šipky směřující od operátora k robotovi do Parameter serveru znázorňují zpětný tok informací o nastaveních ve vizualizačním prostředí a budou vysvětleny níže.



Obrázek 5.5: Nákres pohledového objemu a zarovananého framu z barevné kamery do scény

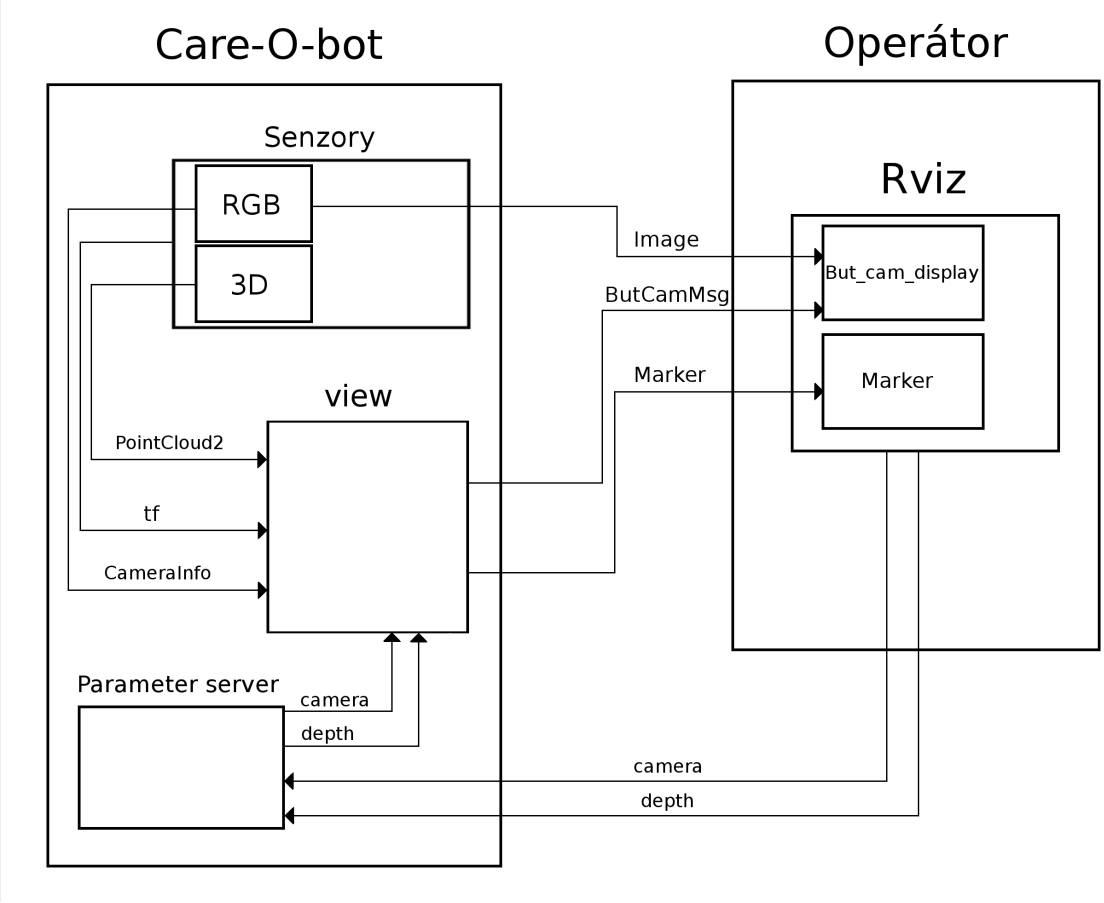
5.1.3 Analýza existujících displejů prostředí Rviz

Již na obrázku 5.6, kde je znázorněn datový tok mezi jednotlivými komponentami vizualizace, je zřejmé, že o zobrazení polygonu otexturovaného aktuálním snímkem kamery bude potřeba vytvořit nový `display` pro vizualizační prostředí Rviz. Principy vytvoření takovéhoto nového displeje budou vysvětleny později v implementační části.

Veškeré displeje prostředí Rviz jsou objekty zděděné z bázového objektu `rviz::Display`. Z něj je třeba odvodit zcela nový display pro potřeby navržené vizualizace. Přestože jsou potřeby tohoto displeje příliš specifické, než aby mohl být použit některý z předdefinovaných displejů prostředí Rviz, jeho podoba bude zároveň dosti podobná některým konkrétním rysům těchto předdefinovaných displejů. Proto se nabízí možnost použít některý z těchto displejů a jeho funkcionalitu pouze upravit tak, aby vyhovovala potřebám nového vizualizačního prvku. Vlastnosti, které jsou potřeba pro nový displej, jsou následující:

- Možnost zobrazit v 3D pohledu prostředí Rviz obdélník viditelný z obou stran.
- Možnost měnit pozici a velikost tohto obdélníku s informacemi publikovanými na zvolených topicích.
- Nový display musí být schopen sám odchytávat snímky z barevné kamery publikované na zadaném topicu.
- Textura zobrazeného obdélníku je dána aktuálním snímkem kamery a je synchronizovaná s polohou obdélníku (tedy aby textura vždy vizualizovala barevný snímek scény viditelné "za"obdélníkem).
- Polygon by měl mít nastavitelnou průhlednost, aby nebránil výhledu na point cloud.

Jak je zřejmé z předchozích kapitol návrhu vizualizace, kdy byl nový display označován jako *marker*, první volbou předdefinovaného displeje se nabízí být display typu Marker (viz



Obrázek 5.6: Propojení a typy zpráv v robotovi a mezi robotem a vizualizačním prostředím

4.2.2). Ten částečně požadované funkce implementuje, ale veliká část zcela chybí a některé by byly relativně komplikovaně implementovatelné.

+ Marker umožňuje zobrazit ve 3D pohledu prostředí Rviz libovolný geometrický tvar.
+ Zároveň je možné měnit pozici a orientaci tohoto tvaru pomocí zpráv ze zvoleného topicu.
– Marker odchytává pouze jeden typ zprávy pro polohu geometrického primitiva, odebírání snímků kamery by bylo třeba přidat.
– Marker neumožňuje přidávat geometrickým prvkům texturu.
– V možnostech úpravy vlastností Markeru také chybí možnost upravit průhlednost.

Při důkladnějším zkoumání jiných typů displejů prostředí Rviz vyšlo najevo, že mnohem lépe potřebám nového displeje vyhovuje pro základ funkcionalita displeje typu `Map`.

+ Map zobrazuje obdélníkový polygon v 3D pohledu prostředí Rviz.
+ Polygon displeje Map má texturu, která je odebírána ze zadaného topicu.
+ Display Map má nastavitelnou průhlednost.

– Map neumožňuje měnit pozici a orientaci polygonu s mapou.

– Zároveň Map neodebírá jiný topic, než zmíněný topic s texturou (tedy se samotnou mapou).

Z předchozích výčtů je zřejmé, že jako základ nového displeje bude lepší použít display typu *Map*.

5.1.4 Očekávaný přínos vizualizace

Vzhledem k dříve popsaným problémům se vzdáleným ovládáním robota v sekci 2.1.1 lze očekávat zlepšení v některých ze zmíněných bodů. Jelikož zorný úhel 3D kamery bývá obvykle větší, než je tomu u běžných barevných kamer, přítmonost point cloudu ve stejném okně s barevným videem částečně pomůže s problémy omezeného zorného úhlu barevné kamery, bez použití modelu okolí z nasnímané hloubkové mapy i v tomto případě bude omezený zorný úhel znatelný problém. Pokud by při vizualizaci bylo zařízeno, že se model robota bude vykreslovat ve skutečné poloze vzhledem k vodorovné hladině (na příklad s použitím gyroskopů, nebo pomocí telemetrie a modelu okolí z hloubkové mapy), může tato vizualizace vyřešit potíže s nesprávným vnímáním polohy a orientace robota. Co však lze říct s jistotou je, že díky zobrazenému point cloudu není možné nesprávně odhadnout hloubku viditelné scény, protože operátor vidí přesný model skutečné scény a zároveň neztratí informace o skutečném vzhledu okolí, zobrazeném na barevných snímcích.

5.2 Texturování point cloudu

Druhou možností řešení sjednocení point cloudu a videa je obarvovat jednotlivé body point cloudu barvami pixelů videa. U této varianty je také možnost zobrazení pohledového tělesa stejně jako u první varianty 5.1. Rozdíl je ale ve vizualizaci barevné informace z kamery. V tomto případě není video zobrazováno na nový polygon ve scéně, ale barva je přímo aplikována na point cloud.

Princip jedné z možných technik použitelných k obarvení point cloudu je popsán v části o PCP algoritmu 2.2.2.

5.2.1 Technické možnosti obarvování point cloudu

Jelikož jsou k dispozici dvě paralelně umístěné kamery, je možné vyhodnocovat barvu každého bodu point cloudu pomocí Point Cloud Painter algoritmu [1] (viz 2.2.2). Vzhledem k tomu, že kamery jsou od sebe vzdálené jen několik centimetrů, je však pravděpodobné, že výsledný rozdíl mezi takto obarveným point cloudem a obarvováním pouze jednou sadou snímků bude zanedbatelný.

U této varianty je očekáváno zlepšení stejných problematických aspektů vzdáleného ovládání robota jako u předchozí varianty popsané výše v odstavci 5.1.4.

Kapitola 6

Implementace vizualizace pro ovládání robota

Obě varianty vizualizace pro ovládání robota s využitím fúze dat z laserového 3D senzoru a barevné kamery byly popsány v předchozí kapitole 5. V následujících sekcích budou podrobněji popsány kroky potřebné k reálné implementaci navržených prvků.

6.1 Zobrazení videa na polygonu

Jak bylo nastíněno v kapitole 5.1, v této variantě bude video z barevné kamery zobrazováno na (poloprůhledném) polygonu před robotem v dobře zvolené vzdálenosti od robota. Volitelně může být v prostoru zvýrazněn také pohledový objem, ohraničující ve scéně vše, co je viditelné danou barevnou kamerou robota.

Nejprve bude přiblížena na první pohled snazší část, tedy zobrazení pohledového objemu, která ve finální verzi bude tvořit relativně samostatnou část. Podle původního návrhu z kapitoly 5.1 je možné těleso znázornit pomocí *markerů*, což je také nejjednodušší způsob. Výpočet všech parametrů pro *marker* je teoreticky možný provádět kompletně na straně vizualizačního prostředí s tím, že potřebné informace získá odebírání zpráv z příslušných topiců. Tím by však vznikl redundantní datový tok mezi robotem a vizualizačním prostředím. Vizualizační prostředí potřebuje pro zobrazení markeru pouze několik málo parametrů, jako jsou pozice a orientace jednotlivých primitiv. Není proto nutné zasílat na stranu operátora všechny parametry kamery, nebo celý point cloud.

6.1.1 Výpočet parametrů pohledového jehlanu

ROS používá k získání snímků z kamery model dírkové kamery 3. Pohledový objem je tedy znázornitelný jehlanem s vrcholem v dírce kamery a směřující od roviny obrazu. Výpočet je proveden v souřadném systému spojeném s kamerou a souřadnice jsou až později převedeny do globálního souřadného systému spojeného se scénou. Rozměry pohledového jehlanu lze spočítat z vnitřních parametrů kamery, které jsou k dispozici ve zprávách typu `sensor_msgs::CameraInfo`. Tyto zprávy jsou zasílány prostředníctvím topiců pro každou kameru zvlášť (levou a pravou barevnou kameru a 3D hloubkovou kameru umístěnou níž mezi nimi). Pro tento účel se není třeba zabývat více kamerama najednou, protože by se pohledové objemy i obrazy kamer překrývaly, což by značně znepřehledňovalo scénu a kromě toho by to žádný užitek nepřineslo. Budou-li výpočty prováděny pro jednu konkrétní kameru, možnost volby této kamery by měla být na straně operátora, tedy ve vizualizačním prostředí.

Jak je vidět v grafu 5.6, představeném v návrhu vizualizace, tok inforamcí od vizualizačního prostředí směrem k robotovi směřuje do tzv. *Parameter serveru*, což je prostředek ROSu pro uchovávání globálních parametrů. Parametr lze snadno nastavit i získaz zpět jeho hodnotu pomocí jednoho z dvojice příkazů:

```
ros::param::set("/param_name", value);
ros::param::get("/param_name", value);
```

Zde `/param_name` je název parametru, pod nímž bude uložen na serveru. Názvy podporují hierarchické členění do *namespaces*, oddělených lomítky. `Value` je samotná hodnota pro vkládání parameru, případně proměnná pro vyzvednutí hodnoty. Parameter server podporuje proměnné typu textový řetězec, logická hodnota, celé číslo a číslo s plovoucí řádovou čárkou. Zvolená kamera je tedy ukládána na parameter server pod názvem `/but_data_fusion/camera` s ohledem na namespace balíčku, je typu textového řetězce a obsahuje typ zvolené kamery z níž si node pro výpočet parametrů pohledového objemu snadno odvodí topic s parametry dané kamery.

Proces tedy odebírá zprávy topicu `/stereo/left/camera_info`, `/stereo/right/camera_info` nebo `/cam3d/depth/camera_info`, jimiž jsou zásílány parametry jednotlivých kamer. V dokumentaci jsou vysvětleny podrobně všechny atributy těchto zpráv [12], pro tuto práci jsou nejdůležitější pole `width` a `height` s šířkou a výškou obrazové roviny dírkové kamery a matice `K` s vnitřními parametry kamery. Matice `K` je blíže popsána v kapitole 3, pro potřeby výpočtu parametrů pohledového objemu stačí vědět, že lze z této matice získat souřadnice dírky kamery a ohniskovou vzdálenost.

S těmito informacemi lze snadno spočítat pozice jednotlivých vrcholů pohledového jehlanu. Princip je znázorněn na obrázku 6.1. Ten zobrazuje, jak je možné spočítat jednu souřadnici (v ose y) horního vrcholu pohledového jehlanu. Obdélník v levé části obrázku znázorňuje dírkovou kameru při pohledu ze strany. Rozměr y je vzdálenost spodní hrany roviny obrazu od hlavní osy kamery. Ten je možné spočítat pomocí výšky obrazové roviny a pozice hlavního bodu kamery (všechny tyto údaje jsou k dispozici ze zprávy *CameraInfo*). Vzdálenost *fy* je ohnisková vzdálenost. α je úhel, který svírá paprsek dopadající na spodní hranu obrazové roviny s hlavní osou kamery a y' je výsledná souřadnice, kterou je třeba spočítat. Vzdálenost *depth* je proměnná podle požadované hloubky pohledového jehlanu. Výpočet této jedné souřadnice řeší rovnice 6.1, 6.2 a 6.3.

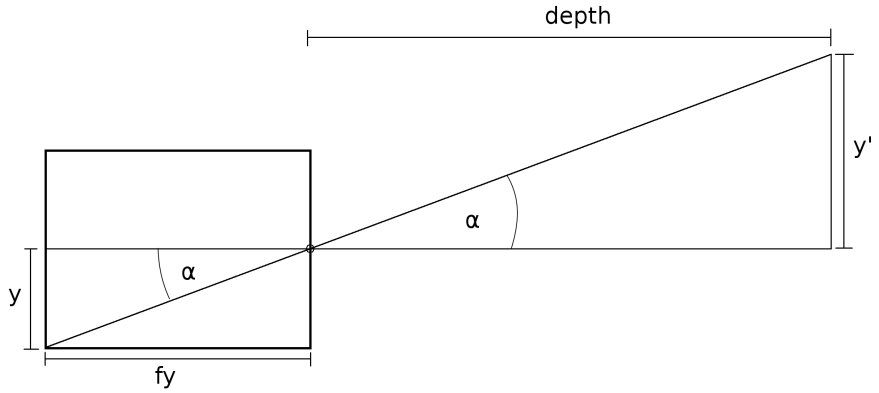
$$tg\alpha = \frac{y'}{depth} \tag{6.1}$$

$$tg\alpha = \frac{y}{f_y} \tag{6.2}$$

$$y' = \frac{y}{f_y} depth \tag{6.3}$$

6.1.2 Zobrazení pohledového objemu

Za předpokladu, že jsou spočítány parametry pohledového jehlanu v souřadném systému se středem v dírce kamery, je třeba ke korektnímu zobrazení pohledového objemu učinit několik kroků. K výpočtu přesných souřadnic vrcholů je nutné zvolit požadovanou hloubku jehlanu. Ta je nastavena tak, aby se dynamicky měnila vzhledem k okolní scéně. Konkrétně je hloubka jehlanu počítána ze vzdálenosti nejvzdálenějšího bodu zobrazovaného point cloudu. Navíc



Obrázek 6.1: Výpočet vrcholů pohledového jehlanu

byla vzhledem k testovacím datům a simulaci robota zvolena maximální hloubka jehlanu `15,0 m`. Nyní, když je určena hloubka jehlanu, lze spočítat přesné souřadnice všech pěti bodů jehlanu. Vrchol bude zřejmě ve středu souřadného systému, jehož počátek je spojen právě s dírkou kamery, zbylé čtyři jsou spočteny postupem popsaným v předchozí sekci 6.1.1. Dalším krokem je převedení souřadnic do globálního souřadného systému, jehož poloha se nemění s časem, tedy do souřadného systému spojeného se scénou. Za tento souřadný systém poslouží rámec `/map`, který je spojen s dvourozměrnou mapou scény, kterou si robot může pomocí balíčku *cob_2dnav* dynamický vytvářet. O obecných principech převodů mezi souřadnými rámci je psáno v kapitole o balíčku TF 4.3.1, jehož je při těchto výpočtech s výhodou používáno. Nyní budou popsány kroky nutné k převodu těchto konkrétních bodů mezi těmito konkrétními souřadnými systémy.

Programově je převod mezi těmito souřadnými systémy proveden pomocí objektu `tf::TransformListener` z balíčku ROSu TF. Jeho metoda `waitForTransform` zastaví na určitý čas běh programu a čeká, dokud není k dispozici transformace (tedy cesta napříč stromem souřadných rámců) mezi zadanými rámci. Mezi důležité parametry volání funkce patrí především zdrojový a cílový souřadný rámec, maximální čas, po který se má na transformaci čekat, a také časová známka okamžiku, pro který chceme transformaci získat. Tato metoda tedy umožňuje nalézt transformaci mezi dvěma rámci i z libovolného času předchozího. Samotné zavolání `waitForTransform` transformaci nezjišťuje, pouze čeká, až bude připravena, aby volání následující metody, konkrétně `lookupTransform`, neskončilo chybou. `lookupTransform` požaduje stejné parametry, jako předchozí metoda, k nimž musí být přidán minimálně jeden navíc, a to proměnná typu `tf::StampedTransform`, do níž má být vyhledaná a vypočtená transformace uložena.

Předtím, než je však možné počítat vrcholy pohledového tělesa a transformovat je do daného souřadného systému, je třeba se zamyslet nad synchronizací všech informací získaných za běhu nodu z topiců. V tomto konkrétním případě si node odebírá pravidelně zprávy týkající se parametrů barevné kamery, informace o stromu souřadných systémů a transformací mezi rámcem barevné kamery a rámcem scény a k tomu všemu zprávy s aktuálním point cloudem snímaným 3D kamerou (pro výpočet vykreslované hloubky pohledového objemu a později i pro výpočet vzdálenosti polygonu se zobrazeným videem). Všechny tyto zprávy jsou časově závislé a je třeba se postarat o to, aby v momentě, co dojde k jejich zpracování, se všechny týkaly stejného (pokud možno co nejaktuálnějšího) okamžiku. V této situaci navíc půjde o dva různé typy synchronizací. První se musí postarat o to, že ke zprávě o parametrech kamery bude k dispozici i strom transformací mezi rámcem

kamery a scény. To řeší třída `tf::MessageFilter`, jejímuž konstruktoru je zadán cílový rámec (pro tuto aplikaci tedy */map*) a stará se o to, aby událost příchodu nové zprávy byla uskutečněna jen v případě, že bude zároveň s ní k dispozici průchod stromem transformací mezi danými dvěma rámci. Druhý typ synchronizace musí zařídit, aby v případě, že bude k dispozici zpráva o parametrech kamery, byla zároveň přijata i zpráva s point cloudem se stejnou časovou známkou (tedy příslušná stejnému okamžiku v čase). Pro tento typ synchronizace je k dispozici třída `message_filters::Synchronizer`. Jde o generickou třídu, která přijímá jako typový parametry synchronizační politiku a jako parametry konstruktoru především listenery jednotlivých zpráv (ať už jde o listener typu `tf::MessageFilter` nebo `message_filters::Subscriber`). Synchronizační politika je v tomto projektu nastavená na typ *ApproximateTime*, která toleruje malý rozdíl časových známek zpráv. Důvodem této tolerance jsou drobné nepřesnosti v časových známkách většiny zpráv. Synchronizační politika typu `ExactTime` zprávy s časovými známkami rozílnými byť jen o desetitisícinu sekundy zahazuje jako nesesynchronizované. S takovouto politikou dorazila až do zpracovávající funkce pouze přibližně jedna z dvaceti zpráv.

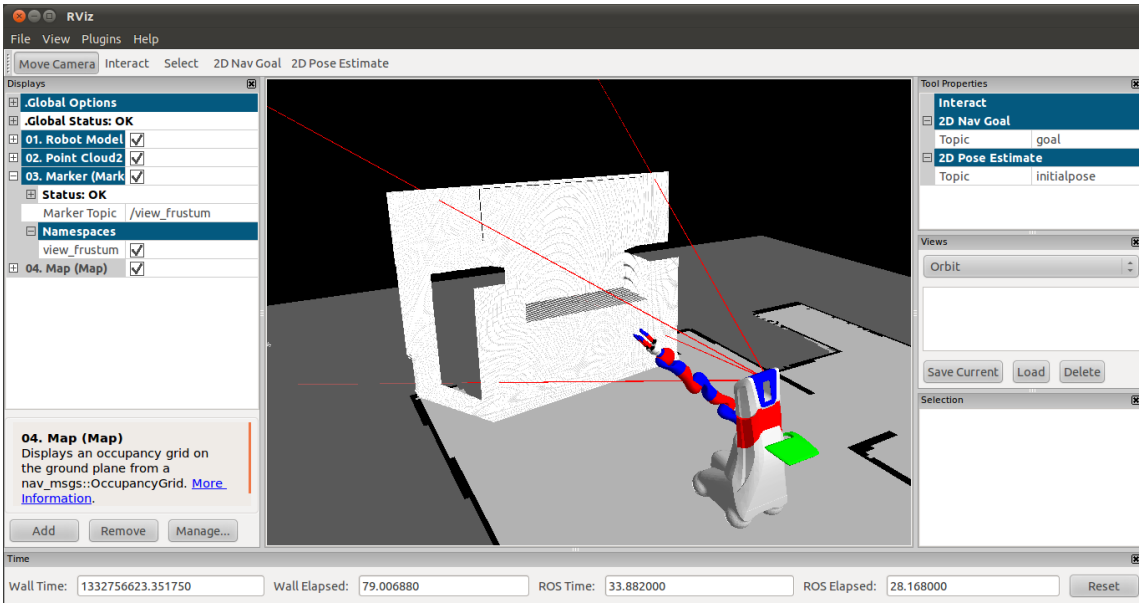
Když jsou vrcholy transformovány do vhodného souřadného systému, je možné jimi vykreslit požadovaný pohledový jehlan do vizualizačního prostředí Rviz 4.2.2. Jak už bylo předloženo v návrhu 5, ideálním prostředkem k zobrazení jednoduchých geometrických primitiv ve vizualizačním prostředí jsou takzvané *markery* 4.2.2. Pro vykreslení sady úseček je k dispozici marker typu `LINE_LIST`, který obashuje pole bodů, v němž jednotlivé dvojice bodů určují samostatné úsečky .

Připravený objekt markeru se z běžícího nodu posílá na zpracování vizualizačnímu prostředí pomocí objektu typu `ros::Publisher`, kterému jsou v konstruktoru předány parametry, určující jaký typ zpráv bude publikovat nebo jaké bude jméno topicu, na který se mají zprávy zasílat. Jsou-li zprávy korektně zasílány na zadaný topic, v prostředí rviz je možné pohledové těleso snadno zobrazit přidáním zobrazování displeje typu marker a zadat mu zvolený topic, v němž informace o geometrii primitiv proudí. Výsledek je ukázán na obrázku 6.2. V centrální části vizualizačního prostředí je zobrazený pohledový objem robota. V levé části v seznamu displejů jsou vidět vlastnosti displeje, který toto těleso reprezentuje. Důležitá je především položka *Marker Topic*, což je název topicu, na něhož dříve popsaný proces publikuje zprávy s geometrií Markeru.

6.1.3 Výpočet parametrů polygonu s obrazem kamery

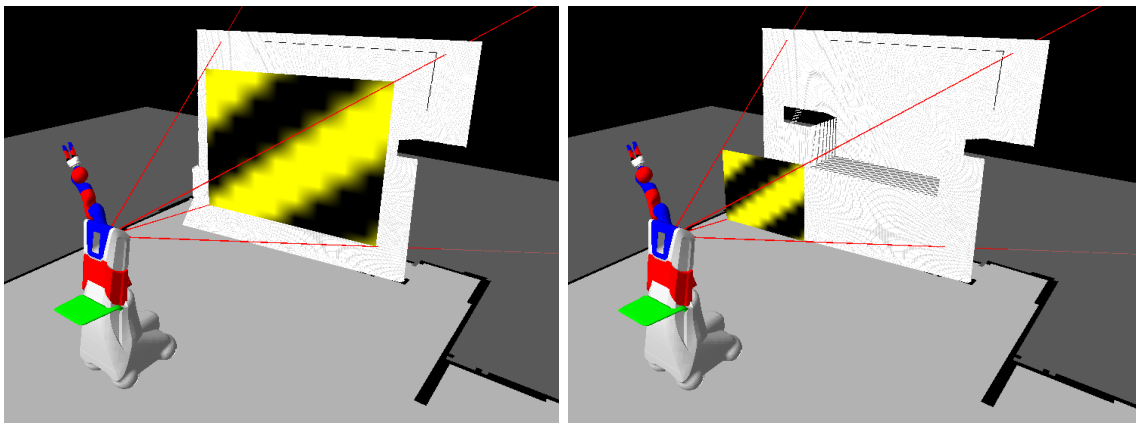
K tomu, aby mohl být zobrazen uvnitř pohledového objemu polygon s aktuálním obrazem kamery, je potřeba několik dodatečných výpočtů. Teoretický základ je připravený z výpočtu parametrů pohledového tělesa v kapitole 6.1.1, v níž bylo popsáno, jak vypočítat především čtyři vzdálenější vrcholy pohledového jehlanu v libovolné hloubce. K určení geometrie požadovaného polygonu bude tento typ výpočtu zcela stačit. Pouze parametr *depth* v tomto případě neznamená hloubku pohledového tělesa, ale vzdálenost od kamery, v níž se má polygon vykreslit. K vyřešení zůstávají dvě otázky - v jaké hloubce polygon vykreslovat a jak informace o jeho geometrii předat vizualizačnímu prostředí.

Co se týká vzdálenosti polygonu od kamery, dobrým řešením se zdá být odvození pozice polygonu od aktuálního point cloudu, s nímž by mohlo teoreticky dojít ke kolizi. Vzdálenost se tedy bude dynamicky měnit s pozicí jednotlivých bodů point cloudu. Aby bylo zajištěno, že nedojde k prolínání vykresleného polygonu se zobrazeným point cloudem, nejjednodušší řešení je, aby se polygon vykresloval vždy blíže kameře, než je nejbližší bod point cloudu. Navíc je v programu ošetřen případ, kdy by i nejbližší bod point cloudu byl kameře velice



Obrázek 6.2: vizualizace pohledového objemu pomocí displeje typu Marker

vzdálený a polygon se záběrem kamery by se tak zobrazil příliš daleko od robota, což by znepříjemňovalo navigaci robota. Je proto nastavena maximální vzdálenost vykreslení polygonu na `15,0 m`. Dále bylo při testování zjištěno, že je někdy vhodné, aby měl operátor sám možnost přizpůsobit vzdálenost vykreslení polygonu podle okolních podmínek, které nelze programově ohlídat. Proto byla displeji v pluginu přidána nastavitelná vlastnost ve formátu desetinného čísla, kterou si může operátor nastavit relativní pozici polygonu mezi maximální vzdáleností a polohou kamery (hodnota 1.0 vykreslí polygon v maximální vypočítané vzdálenosti, 0.5 v poloviční vzdálenosti vzhledem ke kameře a tak dále). Možnosti polohování polygonu jsou znázorněny na obrázku 6.3. Jelikož výpočet pozice polygonu není prováděn na straně operátora, je nutné informaci o požadované vzdálenosti vykreslení předat zpětně na stranu robota. Toho lze docílit na příklad využitím *Parameter serveru*, stejně jako při předávání informace o zvolené kameře (viz 6.1.1).



Obrázek 6.3: Demonstrace renderování polygonu v pohledovém objemu robota s implicitní texturou a se vzdáleností vykreslení nastavnou na hodnotu 1.0 (vlevo) a 0.4 (vpravo)

K předání informací o geometrii polygonu vizualizačnímu prostředí Rviz jsou ideálním prostředkem v tomto případě zprávy (*messages)* proudící skrz topic (viz 4.1). Komplikovanější je však volba typu zasílané zprávy a tedy formát, v jakém se bude informace o geometrii odesílat. K zaslání informací o poloze, orientaci, barvě a podobných vlastnostech geometrických primitiv je určen typ zprávy `visualization_msgs::Marker`. Ten však kromě požadovaných informací obsahuje také typ markeru, akce markeru, životnost a podobně. Aby nedocházelo k zasílání redundantních informací ve zbytečně velké zprávě, byl definován nový typ zprávy, který obsahuje jen nutná data k jednoznačnému určení polohy polygonu. K tomu, aby byl jednoznačně určen obdélník v prostoru, stačí jeden bod, jeho orientace a výška a šířka obdélníku. K zaslání informace o pozici a orientaci bodu je k dispozici přednastavený typ zprávy `geometry_msgs::Pose`. Ta obsahuje atribut typu `Point` nazvaný *position*, tedy pozici bodu v prostoru určenou třemi souřadnicemi v euklidovském prostoru. Druhý atribut je typu `Quaternion` nazvaný *orientation*, který slouží k specifikaci natočení v prostoru, určeného pomocí čtyř hodnot quaternionu. K zaslání informace o měřítku se obvykle používá atribut typu `geometry_msgs::Vector3`, který zcela postačí potřebám zaslání velikosti obdélníků ve dvou směrech. Tyto dva typy (`geometry_msgs::Pose` a `geometry_msgs::Vector3`) byly zapouzdřeny do jednoho typu zprávy nazvaného `srs_ui_but::ButCamMsg`. Definice typu zprávy se provádí velice jednoduše, a to přidáním souboru s názvem zprávy a koncovkou `.msg` do složky `msg/` v kořenovém adresáři balíčku, jemuž nový typ zprávy přísluší. Zpráva se skládá z hierarchicky tvořených prvků, jak bylo popsáno výše, a v definičním souboru jsou tyto prvky zprávy definovány textově pouze výčtem. Definice nového typu zprávy `srs_ui_but::ButCamMsg` je tak krátký, že následuje kompletní podoba souboru `ButCamMsg.msg`:

```
Header header                   # header for time/frame information
geometry_msgs/Pose pose         # Pose of the object
geometry_msgs/Vector3 scale     # Scale of the object 1,1,1 means default
                                  (usually 1 meter square)
```

6.1.4 Vytvoření pluginu pro prostředí Rviz

Jak již bylo naznačeno v předchozím návrhu této metody vizualizace, je třeba vytvořit prostředek pro zobrazení obdélníkového polygonu, který bude samostatně naslouchat na zadaném topicu a obrazem z něj se bude tento polygon dynamicky otexturovávat. Na rozdíl od zobrazení pohledového objemu neposkytuje prostředí Rviz pro tyto účely ideální display, a proto je nutné takovýto display vytvořit. Toho lze dosáhnout vytvořením pluginu pro Rviz, který bude poskytovat prostředí nově implementovaný display.

Plugin pro prostředí Rviz se neliší v mnohém od ostatních balíčků v systému ROS. Jeho vytváření probíhá v několika krocích, prvním z nich je vytvoření balíčku pro ROS se závislostmi na balících `rviz`, `ogre_tools`, `actionlib`, `tf`, `actionlib_msgs`, `geometry_msgs` a `roscpp`.

```
roscreate-pkg jméno rviz ogre_tools actionlib tf actionlib_msgs
geometry_msgs roscpp
```

Dalším krokem je třeba říci kompileru, aby při překladu našel knihovnu `wxWidgets` a tu společně s knihovnou Ogre propojil s výsledným pluginem. Toho lze dosáhnout přidáním následujících řádků do souboru `CMakeLists.txt` nového balíčku.

```
find_package(wxWidgets REQUIRED)
include(${wxWidgets_USE_FILE})
include_directories( ${wxWidgets_INCLUDE_DIRS} )

target_link_libraries(${PROJECT_NAME} ${wxWidgets_LIBRARIES}
                      ${OGRE_LIBRARIES})
```

V předchozí ukázce z kódu je proměnná `PROJECT_NAME` řetězec určující jméno projektu. Do stejného souboru je nutné přidat také řádek, který kompileru říká, že výsledkem překladu bude knihovna. Stejně jako v předchozí ukázce i zde vystupují proměnné, které musí být inicializovány.

```
rosbuild_add_library(${PROJECT_NAME} ${PROJECT_SOURCE_FILES})
```

Proměnná `PROJECT_SOURCE_FILES` je seznam všech zdrojových souborů knihovny. Hlavičkové soubory v adresáři *include* si kompiler najde automaticky. Tato proměnná může být inicializována následovně:

```
set( PROJECT_SOURCE_FILES src/but_data_fusion/but_cam_display.cpp
                          src/but_data_fusion/init.cpp )
```

Z pochopitelných důvodů potřebuje každý plugin zdrojové soubory s implementovanou funkcionalitou (v případě této práce s implementací displeje) a zdrojový soubor pro inicializační metodu. Inicializační metoda se nachází obvykle v souboru `init.cpp`, který se stará o registraci nové třídy v prostředí Rviz. Kód je velmi jednoduchý a o registraci třídy se stará volaní jediné metody.

```
#include "rviz/plugin/type_registry.h"
#include "but_data_fusion/but_cam_display.h"

extern "C" void rvizPluginInit(rviz::TypeRegistry* reg)
{
    reg->registerDisplay<rviz::ButCamDisplay>("CButCamDisplay");
}
```

V ukázce výše jsou zadány konkrétní hodnoty pro cesty k hlavičkovému souboru, řetězec, který bude výsledný display reprezentovat v prostředí Rviz (*CButCamDisplay*) a třídu, která tento display implementuje (`ButCamDisplay`). Tato třída je tedy implementována v zadaném hlavičkovém souboru a příslušném souboru se zdrojovým kódem. V tomto kódu je nutné vytvořit novou třídu, jejíž název je již použit v předchozím kódu, který tuto třídu registroval pro Rviz. Nutností je, aby třída byla potomkem třídy `rviz::Display`. Další nutností kódu je implementace virtuálních metod nadřazené třídy. Jedná se o tyto:

- `virtual void targetFrameChanged()`

  Metoda, která se volá v případě, že ve vizualizačním prostředí byl změněn rámec *Target Frame*.

- `virtual void fixedFrameChanged()`

  Metoda, která se volá v případě, že ve vizualizačním prostředí byl změněn rámec *Fixed Frame*.

- `virtual void update( float wall_dt, float ros_dt )`

  Metoda volána periodicky vizualizačním prostředím pro potřeby překreslení vizualizace. Parametry jsou časové údaje, určující kdy bylo naposledy zavoláno obnovení vizualizace.

- `virtual void onEnable()`

  Metoda zavolaná v případě, že byl display povolen ve vizualizaci (zaškrtávací políčko *Enable* ve vlastnostech displeje v prostředí Rviz).

- `virtual void onDisable()`

  Metoda zavolaná v případě, že byl display zakázán ve vizualizaci (zaškrtávací políčko *Enable* ve vlastnostech displeje v prostředí Rviz).

- `virtual void reset()`

  Metoda volaná pro obnovení původního stavu displeje (obnovení hodnot proměnných a podobně).

- `virtual void createProperties()`

  Metoda zabývající se vytvořením všech nastavitelných i informačních vlastností (*properties*) displeje viditelných a nastavitelných v grafickém rozhraní prostředí Rviz.

Jelikož jádrem všech balíčků v systému ROS jsou soubory `manifest.xml`, je i v tomto případě třeba upravit jej tak, aby odkazoval na popis pluginu. Odkaz na popis se v manifestu vytváří prvkem typu `export`.

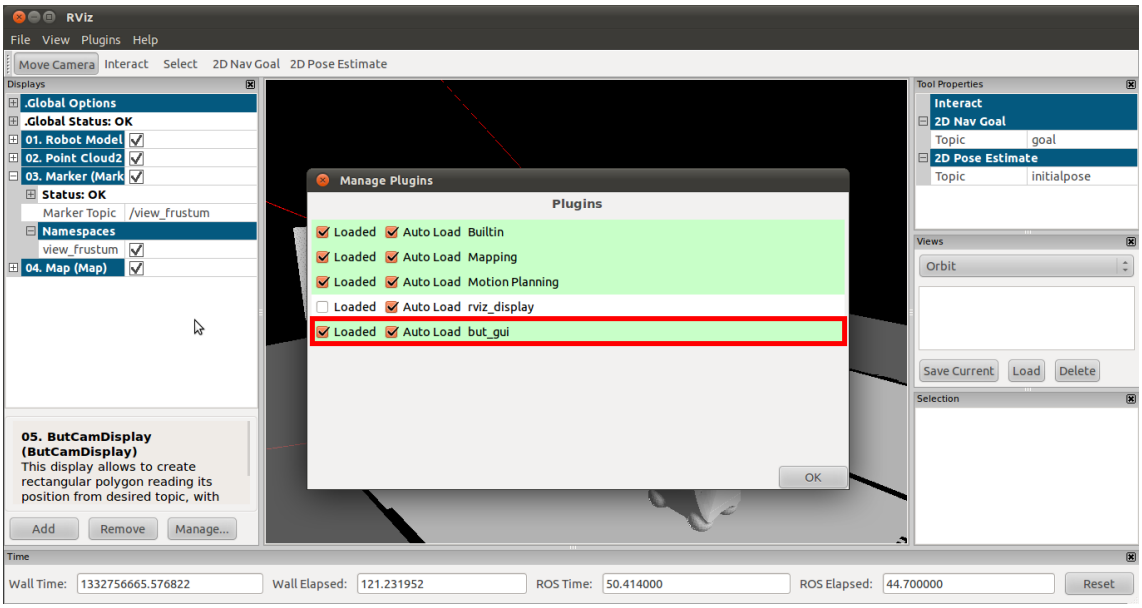
```
<export>
<rviz plugin="${prefix}/lib/but_gui.yaml"/>
</export>
```

V těle prvku `rviz` je odkaz na soubor s koncovkou *yaml*, což je textový soubor obsahující popisek pluginu. Soubor *yaml* má následující formát:

```
name: but_gui
library: but_gui
displays:
  -
    class_name: ButCamDisplay
    display_name: CButCamDisplay
    description: |
      This display allows to create rectangular polygon reading its position
      from desired topic, with dynamic texture
```

Za položkou *name* následuje název pluginu, položka *library* určuje základ názvu výsledné knihovny bez předpony *lib* a bez koncovky *.so*, která se podle předchozí ukázky bude tedy jmenovat `libbut_gui.so`. *class_ name* určuje název třídy implementující display a *display_ name* určuje název displeje v prostředí Rviz. Pod položkou *description* je možné doplnit rozšířený popis displeje. Takovýchto displejů je možné v jednom pluginu implementovat pochopitelně více, v souboru *yaml* by se jejich popis přidal pod návěští *displays* i s pomlčkou.

Toto jsou všechny kroky potřebné k vytvoření pluginu s novým displejem pro vizualizační prostředí Rviz. Po přeložení vytvořeného balíčku příkazem `rosmake` se automaticky vytvoří knihovna s novým displejem a díky registracím v prostředí Rviz bude nyní možné plugin načíst. Na obrázku 6.4 je dialogové okno v prostředí Rviz přístupné z menu položky *Plugins*. V tomto okně je seznam všech dostupných a registrovaných pluginů. U každého z nich lze nastavit, zda má být načten a zda má být načítán automaticky po spuštění. Na obrázku je zvýrazněn červeným rámečkem plugin vytvořený v této práci.



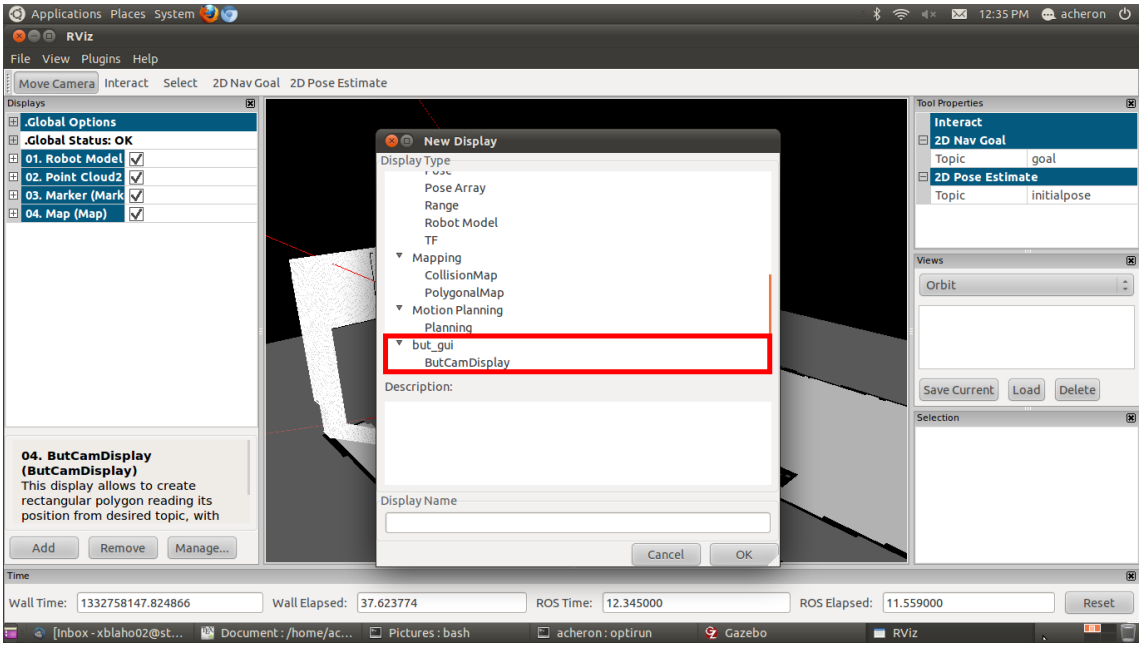
Obrázek 6.4: Dialogové okno načítání pluginů v prostředí Rviz. Červeně zvýrazněný je plugin z této práce

Poté, co je plugin načten, stačí pro jeho přidání do vizualizace kliknout na tlačítko *Add* v levé části prostředí Rviz pod *Display listem* a ve výběru dostupných displejů by měl být k dispozici nově definovaný display. Na obrázku 6.5 je dialogové okno pro výběr displeje, který bude přidán do vizualizace. Červeným rámečkem je zvýrazněný nově vytvořený display ButCamDisplay.

6.1.5 Zobrazení polygonu s obrazem kamery

V této fázi, když je vytvořený prázdný plugin s displejem zděděným od základní třídy *rviz::Display*, popsaný v předchozí kapitole, je potřeba již jen implementovat potřebnou funkcionalitu tohoto displeje. Zároveň byl v kapitole 5.1.3 zvolen vhodný předdefinovaný display `Map`, který poskytuje dostatečnou shodu funkcionality.

Pochoptelně je potřeba udělat řadu úprav. V kapitole 6.1.4 je seznam virtuálních metod,



Obrázek 6.5: Dialogové okno přidání displeje do vizualizace. Červeně zvýrazněný je plugin z této práce

které je třeba displeji implementovat. Většina z těchto metod nevykonává žádnou zásadní práci týkající se funkcionality displeje, proto je možné většinu z nich použít s minimálními změnami z displeje `Map`, z něhož je nový displej odvozen. Větší změny je třeba provést v nastavení takzvaných vlastností (*properties*) displeje. U všech displejů se o nastavitelné vlastnosti stará metoda `createProperties()` (existují i nenastavitelné vlastnosti, které mají pouze informační význam). Jedná se o virtuální metodu a o její spuštění je postaráno v nadřazené třídě `rviz::Display`. V této metodě se nastavejí atributy objektu příslušné jednotlivým vlastnostem (*properties*) displeje. Každá vlastnost má příslušný datový typ, tedy na příklad pro atribut datového typu `float` má jeho příslušná vlastnost displeje datový typ `FloatPropertyWPtr`, atribut typu `string` určující topic systému ROS přísluší vlastnosti typu `ROSTopicStringPropertyWPtr` a podobně. Takto lze displeji nastavit vlastnosti různých datových typů s možností nastavení jejich hodnot samotným uživatelem přes grafické rozhraní prostředí Rviz.

Kromě definice jednotlivých vlastností je nutné tyto proměnné nainicializovat. Toho lze docílit pomocí objektu `PropertyManager` (opět zděděného z nadřazené třídy) zavoláním jeho metody `createProperty`. Jde o generickou metodu, která jako typový parametr očekává datový typ dané vlastnosti. Mezi důležité parametry metody patří řetězec reprezentující název vlastnosti, pod nímž se bude ve vizualizačním prostředí zobrazovat, a ukazatele na metody typu *get* a *set*, které se volají automaticky při změně hodnoty ve vizualizačním prostředí nebo pro získání hodnoty.

Pro vytvořený vizualizační plugin jsou nejdůležitější vlastnosti, do nichž uživatel nastavuje topiky, z nichž budou odebírány informace o pozici a textuře vykreslovaného polygonu. Díky metodám typu *set* těchto vlastností je možné jejich nastavení uživatelem provázat s funkcema, které nastaví odebírání zadaných zpráv, jejich synchronizaci a následné volání callback funkcí, které zařídí pravidelné překreslování polygonu a jeho textury. Odebírání a synchronizace zpráv je prováděna velice podobně jako v případě nodu, provádějícího výpočty

na straně robota (viz odstavec 6.1.2).

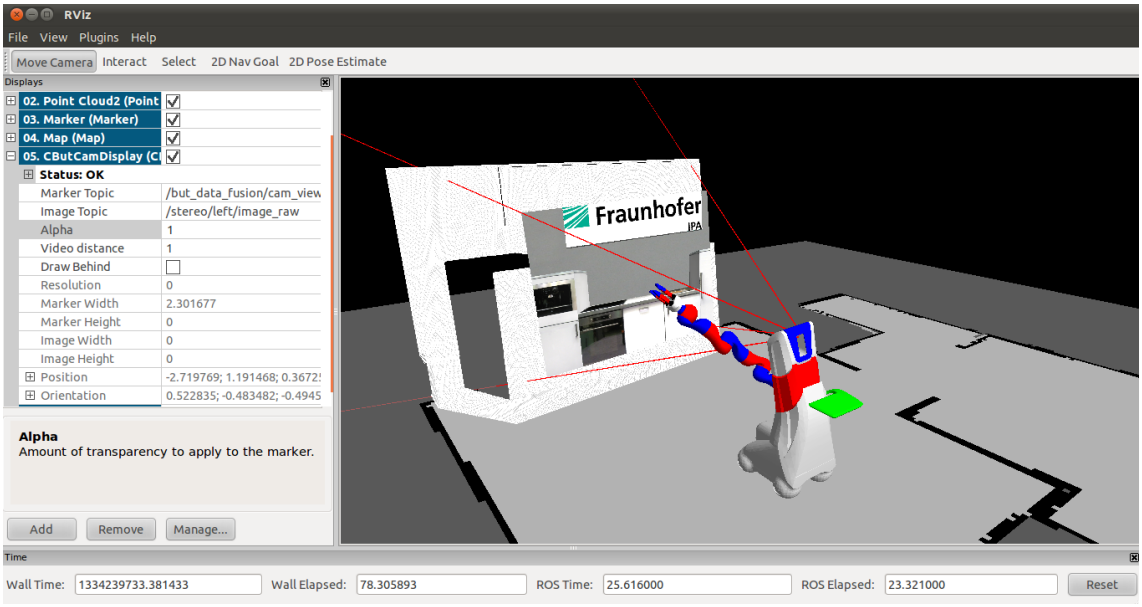
Po příchodu synchronizovaných zpráv s geometrií i texturou polygonu je z callback funkce zavolaná metoda load, které je jako parametr předána zpráva o geomterii polygonu (tedy konstanta typu `srs_ui_but::ButCamMsg` (viz 6.1.3) a metoda loadImage, která dostává zprávu se synchronizovaným snímkem kamery (tedy zprávu `sensor_msgs::Image`).

Plugin vykresluje do vizualizačního prostředí pomocí knihovny Ogre (viz 4.3.2). Jelikož je nový displej potomkem třídy `Display`, která je přímo určena k vykreslování do vizualizačního okna prostředí Rviz, je z této třídy zděděn také ukazatel na prvek typu `Ogre::SceneManager`. Pomocí tohoto objektu lze snadno vytvořit nový uzel scény, k němuž je možné připojit objekty, které je třeba přidat do scény (metodou `attachObject`). Objekt požadovaného obdélníkového tvaru lze vytvořit pomocí objektu `Ogre::ManualObject`. Ten je inicializován mezi voláním své metody `begin` a `end`, přičemž `begin` dostává jako parametry materiál vytvářeného objektu (objekt `Ogre::MaterialPtr`, v tomto případě prozatím prázdný materiál, kterému bude posléze připojena textura) a typ primitiv, z nichž se bude objekt skládat (v tomto případě seznam trojúhelníků, přesněji dvou trojúhelníků typu `Ogre::RenderOperation::OT_TRIANGLE_LIST`). Seznamu trojúhelníků je každý další vrchol přidáván trojicí metod určující pozici, normálu a texturovací souřadnice bodu. V této fázi lze s výhodou využít toho, že ve zprávě s geometrií polygonu je informace jak o pozici polygonu, tak o jeho orientaci v prostoru i rozměrech. Díky tomu je možné polygon vytvořit v počátku souřadnic pouze jako jednotkový čtvercový polygon a jeho geometrii poté upravit voláním trojice metod `setPosition`, `setOrientation` a `setScale`, kterým jsou předány informace z obdržené zprávy. Po každé nově obdržené zprávě pak stačí již jen znovu zavolat tyto transformační funkce, čímž je zařízena vždy správná pozice a orientace polygonu vzhledem k pohybu robota. Zobrazený a korektně umístěný polygon je vyobrazený na obrázku 6.3.

Metoda `loadImage` obstarává korektní texturování polygonu aktuálním synchronizovaným snímkem kamery, který tato metoda obdrží ve svém parametru ve formátu ROS zprávy `sensor_msgs::Image`. V této metodě je vytvořena nová textura (tedy objekt typu `Ogre::TexturePtr`), která je naplněna daty ze zprávy s obrazem kamery. Tato textura je následně napojena na materiál, který byl nastaven polygonu při jeho vytváření. Je třeba si uvědomit, že systém Ogre spravuje textury pomocí takzvaného `TextureManageru`, který obstarává, aby žádná textura nemusela být načítána vícekrát a byla vždy k dispozici. Při dynamickém texturování však každý nový snímek kamery vytváří novou texturu a ukládá ji do paměti. Je proto nutné předchozí textury z `TextureManageru` promazávat, aby nedocházelo k chybám způsobeným zahlcením paměti. Kompletní verze této vizualizace je znázorněn na obrázku 6.6. Více ukázek finální podoby je v pozdější kapitole 7.

6.2 Texturování point cloudu

Oproti předchozí variantě vizualizace, kde implementace přesně odpovídá návrhu z předchozí kapitoly, se bude v tomto případě implementace lišit od původních záměrů. V současné době systém ROS při simulaci Care-O-bota umožňuje zobrazovat point cloud již obarvený samotnou informací ze zařízení *Kinect*, které tak samo provádí fůzi získané hloubkové a barevné informace. Výsledný obarvený point cloud je vyobrazený na snímku 5.2 v návrhu vizualizace. Původní plán byl tedy změněn z obarvování point cloudu na obarvování environmentálního modelu, který je v rámci projektu *SRS* vyvíjen.



Obrázek 6.6: Finální podoba první varianty vizualizace s vykreslovanou vzdáleností nastavenou na 1.0 bez průhlednosti

6.2.1 Environmentální model

Model prostředí je vytvářen ze snímků point cloudu pořízených *kinectem*. Modul nazván `srs_env_model` si vytváří strukturu tzv. *oktomapu*, v níž registruje všechny doposud zaznamenané snímky point cloudu do jediného point cloudu. Výstup původního nebarevného výstupu této oktomapy je na obrázku 6.7 vzniklém otočením robota kolem dokola a poskládáním hloubkové informace o jaho okolí do jednoho modelu prostředí.

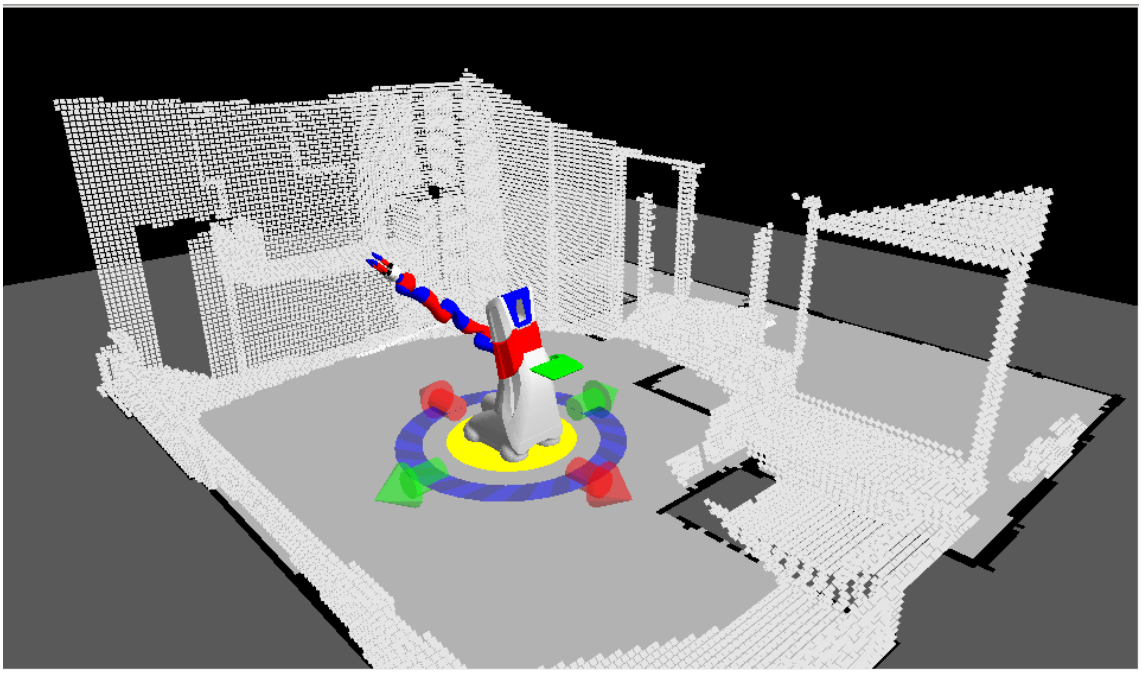
Pro uchovávání nejen informace o pozici, ale také o barvě jednotlivých bodů environmentálního modelu je nutné udělat několik změn v implementaci použité oktomapy a použitých uzlů oktomapy.

6.2.2 Úprava uzlů oktomapy

V prvé řadě je třeba, aby jednotlivé uzly oktomapy byly schopny nést informaci o své barvě. Definice jak jednotlivých uzlů oktomapy, tak celé oktomapy z nich složené je v souborech `include/but_server/octomap.h` a `src/but_server/octomap.cpp`. Po přidání atributů pro uchování barvy v definici uzlů oktomapy (tedy v třídě `EModelTreeNode`) a metod pro ukládání a čtení této barvy jsou jednotlivé uzly schopné nést informaci o své barvě případně alfa kanálu.

Dále je pro budoucí použití potřeba metoda, která zjistí, zda byla barva již dříve uzlu přiřazena. V tomto místě je využito toho, že v konstruktoru uzlů je implicitní barva nastavená na čistě bílou (`#FFFFFF`) a je velice nepravděpodobné, že by v budoucnu byla čistě bílá někdy znovu některému z uzlů přiřazena. Proto se zjišťování, zda je barva implicitní či již nastavená, provádí pomocí porovnání s bílou barvou.

Další metody, které jsou v původní podobě nedostačující (protože se nezabývají barvou uzlů), jsou prostředky pro takzvané *prořezávání uzlů* (*node pruning*) a *expandování uzlů* (*node expanding*). Jde o dvě v podstatě inverzní operace. Prořezávání v případě, že mohou být všichni potomci uzlu sjednoceni do nadřazeného uzlu, smaže tyto potomky a patřičně



Obrázek 6.7: Původní podoba oktomapy bez barevné informace

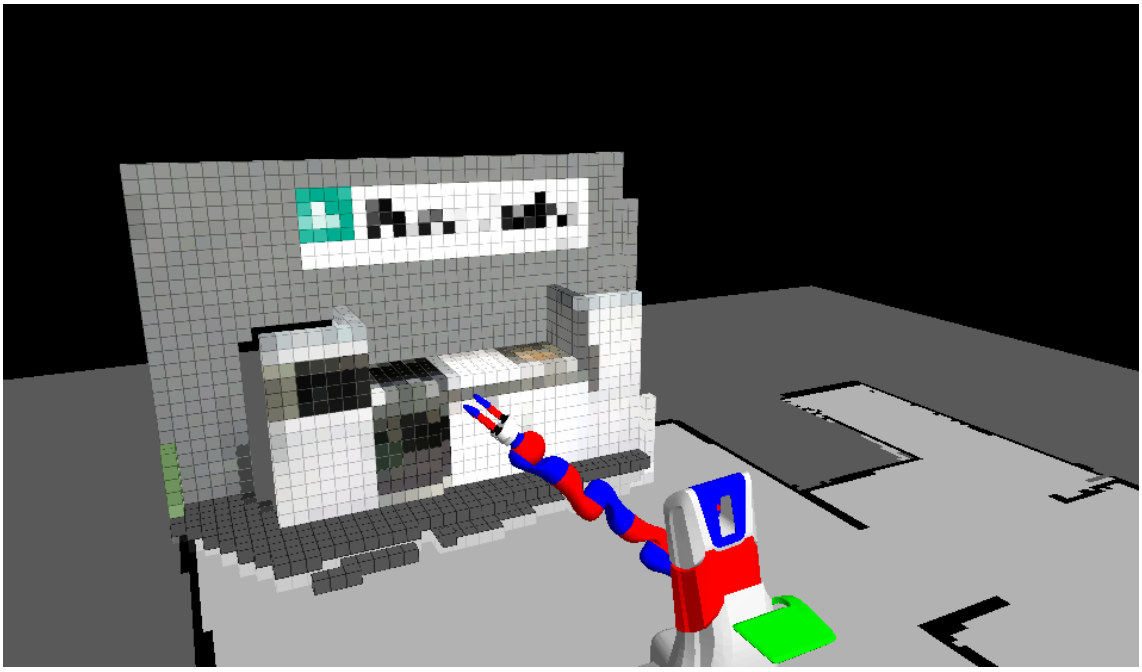
upraví pravděpodobnostní parametr uzlu. Pro zachování barevné konzistence je přidána část kódu, která při prořezávání nastaví barvu nadřazeného uzlu zprůměrovanou barvou všech jeho zrušených potomků. Oproti tomu expandování jednomu uzlu vytvoří osm potomků se stejnou hodnotou pravděpodobnosti, jako má nadřazený uzel. Pro potřeby barevné oktomapy je při této operaci nově vytvořeným uzlům kopírována také barva nadřazeného uzlu.

6.2.3 Úprava oktomapy

Pro implementaci oktomapy je důležitá funkce, která na zadané pozici v oktomapě upraví barvu cílového uzlu s použitím nově příchozí barevné informace. Vyhledávání pozice v oktomapě je implementováno pomocí takzvaných *klíčů*, které lze vygenerovat z 3D souřadnic bodu. Jedinou otázkou tedy zůstává, jakým způsobem integrovat nově příchozí barvu s již dříve definovanou barvou uzlu. Byly vytvořeny dvě metody, každá implementující odlišný přístup slučování barev.

Metoda `averageNodeColor` najde uzel, načte jeho původní barvu a tu jednoduše zprůměruje s nově příchozí barvou. Při použití této metody oktomapa velice rychle reaguje na náhlé změny barvy v okolní scéně, ale stejně rychle propaguje na výstup případné chyby senzorů. Metoda `integrateNodeColor` z nalezeného uzlu slučuje barvu s nově příchozí barvou s přihlédnutím k pravděpodobnostnímu parametru vyhledaného uzlu. Přesněji původní barva se uplatňuje v nové barvě se stejnou váhou, jaká byla logaritmická pravděpodobnost uzlu oktomapy. Nová barva se uplatňuje s pravděpodobností převrácenou. Při použití této metody se tedy scéna postupně s tím, jak přicházejí nové snímky point cloudu, překresluje, dokud nedojde do jakéhosi ustáleného stavu. Stejně tak změny ve scéně se prosazují ve výsledné oktomapě postupně. Porovnání použití různých metod integrace nově příchozích barev je viditelné na snímcích 6.8 a 6.9

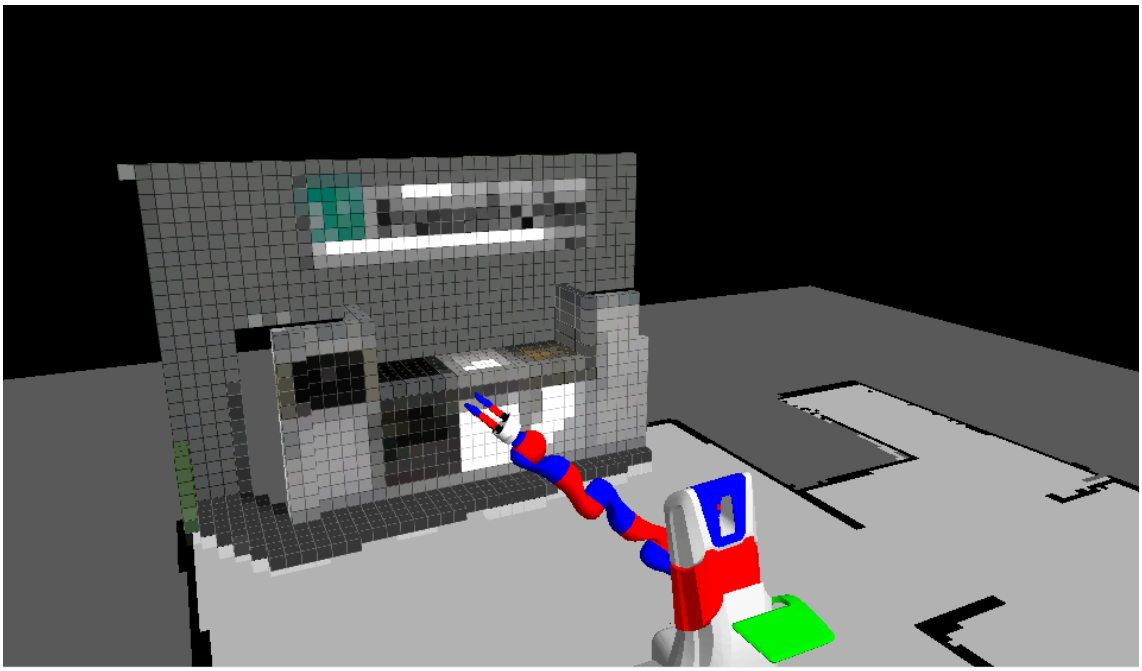
Poslední důležitou součástí barevné oktomapy je metoda, která má na svědomí vkládání



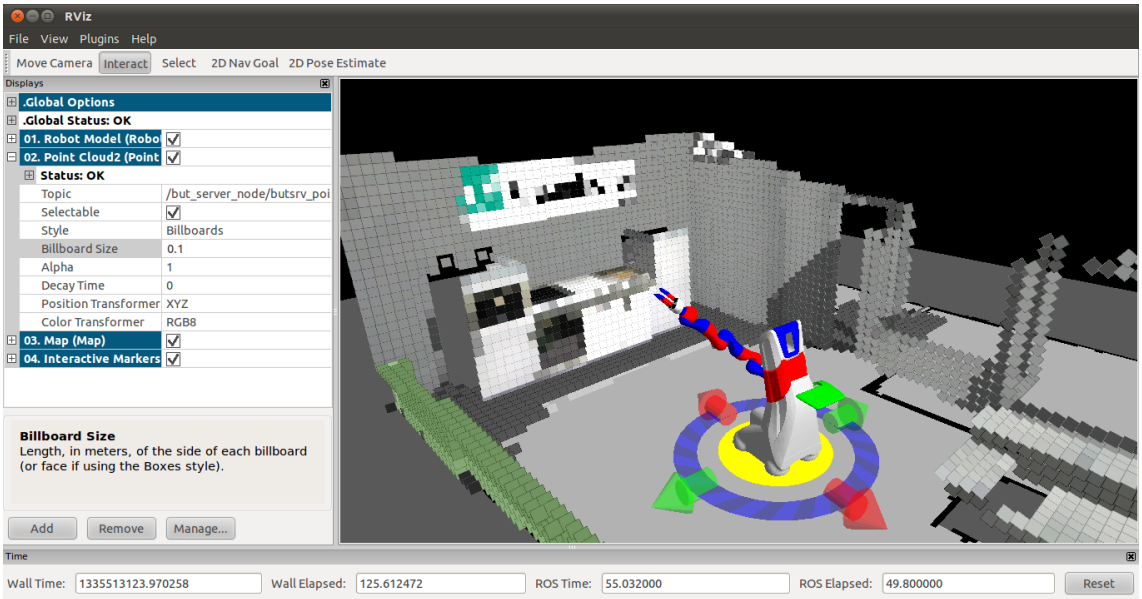
Obrázek 6.8: Obarvená oktomapa s použitím průměrování k integrování nových barev

nových skenů do existující oktomapy. O tuto činnost se obecně stará metoda `insertScan`, která na svém vstupu přijímá point cloud datového typu `octomap::Pointcloud`, což je point cloud bez barevné informace. Proto bylo třeba pozměnit jak implementaci funkce za vzniku nové metody `insertColoredScan`, tak její volání, aby obdržela barevný point cloud. Většina funkcionality je převzata z původní funkce, nicméně je zde přidán kód pro ukládání barvy do jednotlivých upravovaných uzlů. Právě v tomto místě dochází k volbě mezi jednotlivými algoritmy integrace nové barvy v uzlu - tedy volbě mezi metodou `averageNodeColor` a `integrateNodeColor`.

V závěru je třeba upravit několik drobností, jako nastavit odebírání správného tedy obarveného point cloudu ze senzoru (konkrétně se jedná o topic `/cam3d/rgb/points`) a datové typy průchozích point cloudů. Finální podoba obarvené oktomapy vzniklé otočením robota kolem dokola je na snímku 6.10.



Obrázek 6.9: Obarvená oktomapa s použitím pravděpodobností k integrování nových barev



Obrázek 6.10: Obarvená oktomapa

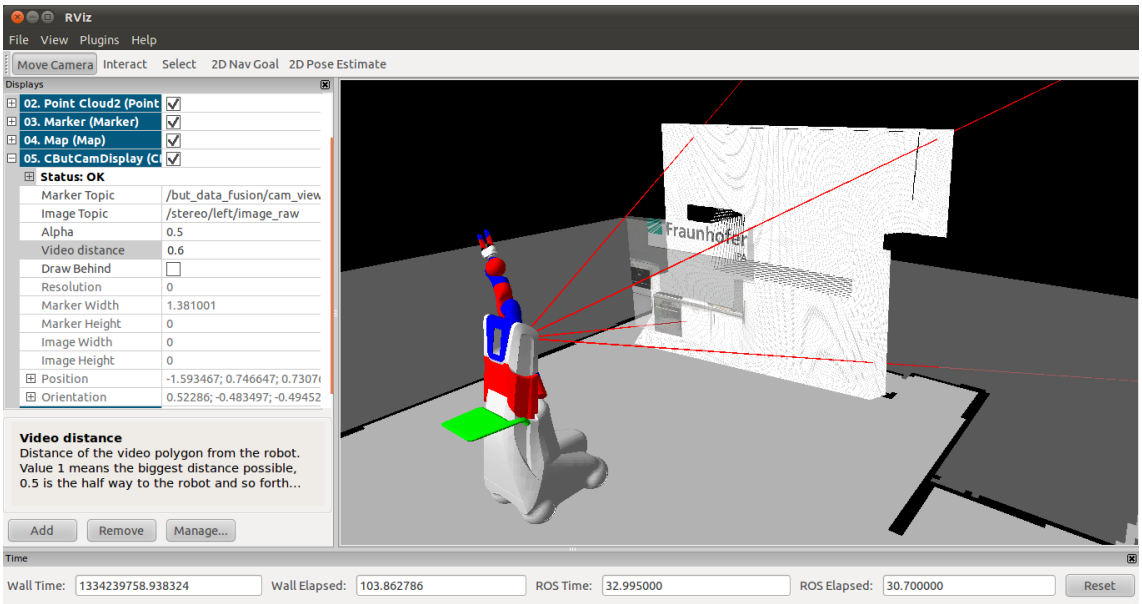
Kapitola 7

Výsledky

V této kapitole jsou shrnuty výsledky praktické implementace navržených vizualizací. Bude popsáno, jak dobře fungují jednotlivé součásti a do jaké míry byla splněna očekávání zlepšení vnímání okolní scény robota s použitím nových vizualizací.

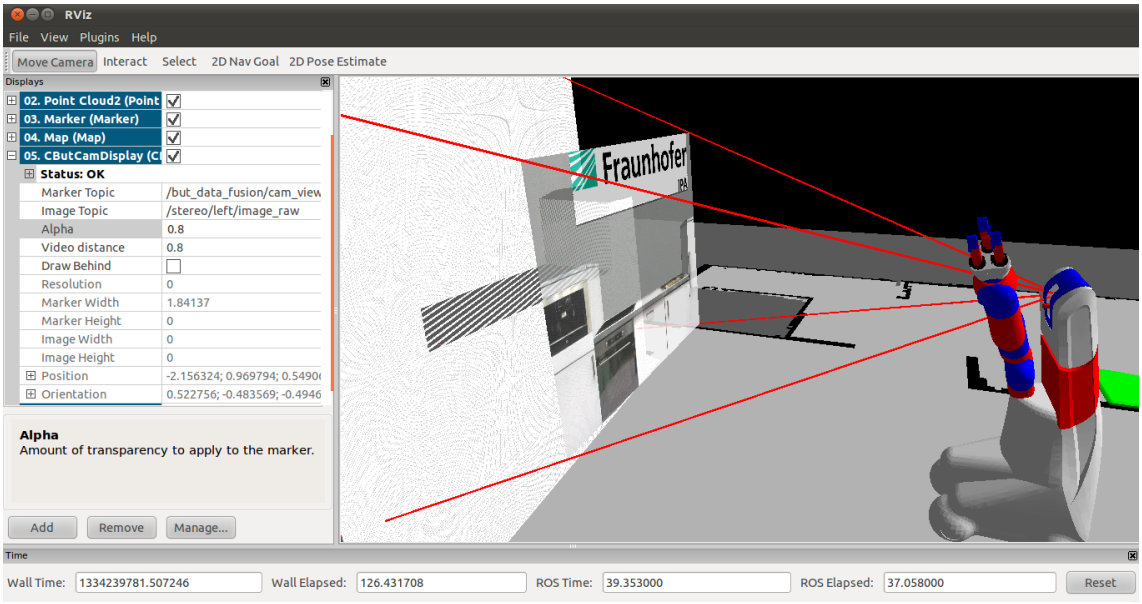
7.1 Zobrazení videa na polygonu

Tato první navržená vizualizace okolí robota podle návrhu sjednocuje informaci o barvě scény a její hloubce reprezentované point cloudem jeho překrytím barevným snímkem. Vzhled vizualizace vypadá podle očekávání, jedna z ukázek byla představena již v předchozí kapitole (obrázek 6.6). Více ukázek s různým nastavením parametrů průhlednosti a pozice polygonu s barevným snímkem je na obrázcích 7.1 a 7.2.



Obrázek 7.1: Finální podoba první varianty vizualizace s vykreslovanou vzdáleností nastavenou na 0.6 a 50% průhledností

U poslední verze této vizualizace byla zároveň změřena obnovovací frekvence, která je společná jak pro obnovování pozice, tak textury polygonu, z důvodu synchronizace těchto



Obrázek 7.2: Finální podoba první varianty vizualizace s vykreslovanou vzdáleností nastavenou na 0.8 a 80% průhledností

Skutečná obnovovací frekvence	Simulační obnovovací frekvence
6.05712 Hz	2.11988 Hz

Tabulka 7.1: Naměřené obnovovací frekvence vizualizace

dvou typů informace. Při měření byla průměrná rychlost simulace 0.34-násobek skutečného času a výsledky jsou průměrem hodnot ze 150 snímků. Výsledky jsou shrnuty v tabulce 7.1.

Z výsledku je zřejmé, že při rychlosti simulace, v jaké byla vizualizace testována, se bude pohledové těleso s texturovaným polygonem pohybovat velice neplynule. V reálném využití s frevencí přes šest snímků za vteřinu již výsledek bude vypadat o poznání lépe. Přesto bylo zmíněno v kapitole týkající se problémů se vzdáleným ovládáním robotů 2.1, že je v praxi pro navigaci robota nezbytná minimální frekvence senzorických informací 10 Hz. Takovéto rychlosti vizualizace nedosahuje a proto bude zřejmě nedostačujícím zdrojem vizuální informace pro manipulaci s robotem. Zlepšení, které tato vizualizace přináší, se týká vnímání okolí robota operátorem, který se díky tomu může lépe zorientovat ve scéně.

Budoucí zlepšení funkčnosti programu by se mohlo týkat zvýšení obnovovací frekvence obrazu vizualizace. V prvé řadě je zde omezení frekvencí samotných zdrojových informací, tedy point cloudu a barevných snímků videokamery, případné zlepšení se tedy bude muset týkat lepší synchronizace těchto informací.

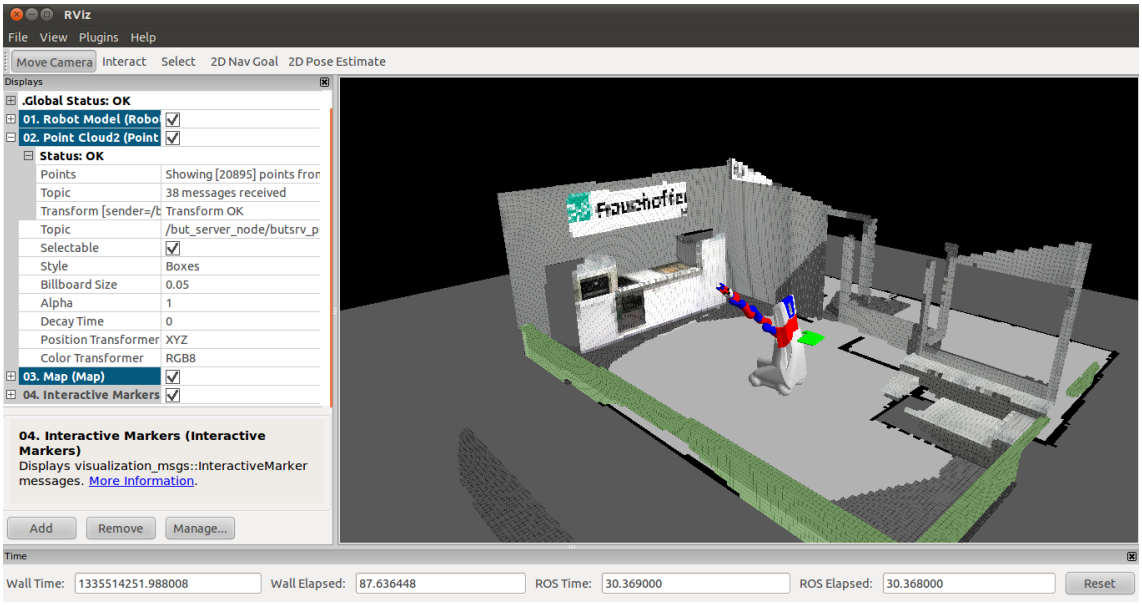
7.2 Texturování point cloudu

Tato varianta oproti návrhu neobarvuje samotný výstup z laserového senzoru, ale umožňuje obarvování environmentálního modelu, který z toho výstupu vychází. I v tomto případě vypadá výsledek dle očekávání. V této variantě navíc jde o opravdovou fůzi hloubkové a barevné informace, kde si tyto informace vzájemně nestíní, na rozdíl od předchozí varianty. První výstup z hotové vizualizace byl představen v předchozí kapitole na snímku 6.10.

Rozlišení oktomapy	Barevná oktomapa	Nebarvená oktomapa
0.5	7.08304 Hz	8.81438 Hz
0.1	2.59516 Hz	2.79103 Hz
0.05	1.4592 Hz	1.46624 Hz

Tabulka 7.2: Naměřené obnovovací frekvence oktomapy

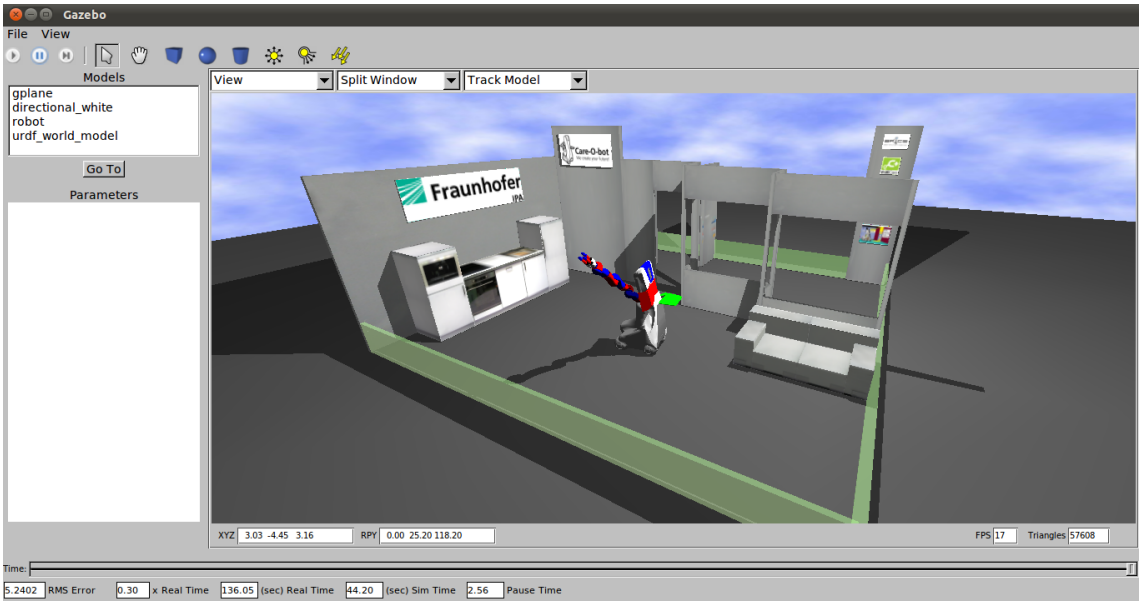
Další ukázkou možností barevné oktomapy je snímek 7.3 pořízený se zvýšeným rozlišením oktomapy v porovnání s opravdovou podobou simulované scény na snímku 7.4.



Obrázek 7.3: Obarvená oktomapa se zvýšeným rozlišením

Zároveň byla provedena měření obnovovací frekvence snímků oktomapy v různých rozlišeních a to jak u barevné, tak nebarvené varianty. Výsledky měření shrnuje tabulka 7.2. Frekvence se vztahuje na reálný, nikoliv simulační čas.

Z naměřených výsledků vyplývá, že obarvování oktomapy má jistý negativní vliv na její rychlost vytváření a publikování. Tento vliv je však nepatrný vzhledem k naměřeným rychlostem u běžně používaného rozlišení (pro většinu navigačních úkolů a u většiny testů bylo používáno rozlišení přibližně `0.1`, kde obarvování způsobuje zpomalení o necelé dvě desetiny snímku za sekundu). Ze snímků je zřejmé, že při tomto způsobu vizualizace bude zřetelně posílena orientace operátora ve scéně, v níž se robot pohybuje. Barevná informace se vzájemně překrývá s hloubkovou informací, ale vzájemně si nestíní. Oprátor by neměl mít potíže s vnímáním polohy a orientace robota či se špatným vnímáním hloubky scény. Rychlost překreslování a vytváření oktomapy je zde úměrná rozlišení oktomapy, což je v podstatě jediné omezení této vizualizace.



Obrázek 7.4: Skutečná podoba simulované scény

Kapitola 8

Závěr

Vzdálené ovládání robota bývá často umožňeno vizualizací okolí robota pomocí video kamery nebo pomocí 3D laserového skenu. Pro lepší orientaci vzdáleného operátora v okolí robota je možnost sjednotit výstupy těchto dvou senzorů do jediného renderovacího okna.

Cílem této práce bylo navrhnout, implementovat a otestovat způsoby fúze hloubkové a barevné senzorické informace do jediného obrazu pro použití při vzdáleném ovládání robotů.

Byly navrženy dva typy vizualizace okolí robota s použitím různých technik datové fúze. V první variantě byla barevná informace vykreslována na poloprůhledném polygonu před hloubkovou informací, zobrazenou pomocí point cloudu. Tato varianta trpí slabou obnovovací frekvencí vykreslování polygonu s obrazem videa, způsobenou snahou o co nejlepší synchronizaci informace o orientaci robota a aktuality snímku barevné kamery. Druhá varianta využívá environmentální model v podobě oktomapy a vzniklý point cloud přímo obarvuje barvou z barevného senzoru. Tato varianta výborně sjednocuje hloubkovou a barevnou informaci o okolí robota. Vytvoření a udržování environmentálního modelu je sice výpočetně mnohem náročnější, než provoz předchozí varianty, ale výsledek je relativně přesný model prostředí, v němž robot existuje. Každá varianta má tedy své výhody i nevýhody, obě však pomáhají lepší orientaci vzdáleného operátora v okolí robota.

Lze očekávat praktické využití kteréhokoliv z navrhovaných řešení při vzdáleném ovládání robotů.

Literatura

[1] A. Abdelhafiz: *Integrating Digital Photogrammetry and Terrestrial Laser Scanning*. Dizertační práce, Bayerischen Akademie der Wissenschaften, 2009.

[2] A. Abdelhafiz, B. Riedel, W. Niemeier: Towards a 3D true colored space by the fusion of laser scanner point cloud and digital photos. In *Proceedings of the ISPRS Working Group V/4 Workshop (3D-ARCH)*, 2005.

[3] A. Abdelhafiz, W. Niemeier: Developed Technique for Automatic Point Cloud Texturing Using Multi Images Applied to a Complex Site. *International Archives of Photogrammetry and Remote Sensing*, ročník 36, č. 5, Září, 2006.

[4] D. J. Bruemmer, D. A. Few, R. L. Boring, J. L. Marble, M. C. Walton, and C. W. Nielsen: Shared Understanding for Collaborative Control. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, ročník 35, č. 4, Červenec, 2005.

[5] J. L. Burke, R. R. Murphy, E. Rogers, V. J. Lumelsky, and J.Scholtz: Final report for the DARPA/NSF interdisciplinary study on human?robot interaction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, ročník 34, č. 2, Květen, 2004.

[6] J. L. Marble, D. J. Bruemmer, and D. A. Few: Lessons learned from usability tests with a collaborative cognitive workspace for human-robot teams. In *IEEE International Conference on Systems, Man, and Cybernetics*, 2003.

[7] J. Manuel: Do You Know? : Robots, How Far Have They Come? `http://simple-article10.blogspot.com/2010/09/robots-how-far-have-they-come.html`, 2010 [cit. 2012-03-20].

[8] J. Y. C. Chen, E. C. Haas, and M. J. Barnes: Human Performance Issues and User Interface Design for Teleoperated Robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, ročník 37, č. 6, Listopad, 2007.

[9] Kolektiv autorů: *Gazebo documentation*. 2007-08-04 [cit. 2011-12-29].

[10] Kolektiv autorů: Care-O-bot - Fraunhofer IPA. `http://www.care-o-bot.de/english/`, 2009 [cit. 2012-03-02].

[11] Kolektiv autorů: The Player Project. `http://playerstage.sourceforge.net/`, 2010-11-26 [cit. 2011-12-27].

[12] Kolektiv autorů: *ROS documentation*. 2011-12-26 [cit. 2011-12-27].

[13] Kolektiv autorů: Microsoft Robotics Studio. `http://msdn.microsoft.com/en-us/robotics/default.aspx`, 2011 [cit. 2011-12-27].

[14] Kolektiv autorů: *OGRE*. 2012-03-18 [cit. 2012-03-20].

[15] Kolektiv autorů: Shadow Robotic System. `http://srs-project.eu/`, 2012 [cit. 2012-03-02].

[16] L. Lart: Larry's Work : Robotic Arm Hardware - CH.ORG II. `http://larryo.org/work/robotics/robotics_hardware.html`, [cit. 2012-03-20].

[17] M. Hanlon: Gizmag : The Care-O-bot 3 - always at your service. `http://www.gizmag.com/the-care-o-bot-3-always-at-your-service/9628`, 2008 [cit. 2012-03-14].

[18] M. K. Sivaraman: *Virtual reality based multi-modal teleoperation using mixed autonomy*. Dizertační práce, Iowa State University, 2009.

[19] M. Y. Yang, Y. Cao, J. McDonald: Fusion of camera images and laser scans for wide baseline 3D scene alignment in urban environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2011.

[20] V. Popescu, P. Rosen, L. Arns, X. Tricoche, C. Wyman, C. M. Hoffmann: The General Pinhole Camera: Effective and Efficient Nonuniform Sampling for Visualization. *IEEE Transactions on Visualization and Computer Graphics*, ročník 16, č. 5, Září/Říjen, 2010.

[21] W. Zhao, D. Nister, S. Hsu: Alignment of Continuous Video onto 3D Point Clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 27, č. 8, Srpen, 2005.

[22] Y. Morvan: *Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video*. Dizertační práce, Eindhoven University of Technology, 2009.

Příloha A

Obsah CD

1. Adresář `src` obsahující zdrojové kódy potřebné ke spuštění programové části práce
2. Adresář `doc` se zdrojovými soubory této teoretické části práce
3. Soubor plakat.png s plakátem prezentujícím tuto diplomovou práci

Příloha B

Manuál

V této části bude definováno, které zdrojové soubory nebo jejich části byly vytvořeny případně změněny v rámci této práce. Dále budou popsány kroky nutné ke spuštění programové části práce.

B.0.1 Zdrojové soubory

Oficiální manuálové stránky zdrojových souborů v rámci SRS projektu se nacházejí na internetové adrese `http://ros.org/wiki/srs_public`, kde lze nalézt podrobný popis veškerých balíčků projektu SRS, tedy i těch vzniklých v rámci této práce.

V Adresáři `src/` jsou zdrojové soubory veškerých balíčků potřebných ke spuštění programové části. Tato práce má na svědomí pouze část z nich, jejichž výčet následuje.

Nově vytvořené soubory

- srs_ui_but/src/but_data_fusion/but_cam_display.cpp
- srs_ui_but/src/but_data_fusion/view.cpp
- srs_ui_but/include/but_data_fusion/but_cam_display.h
- srs_ui_but/include/but_data_fusion/topics_list.h

Nově vytvořené metody v existujících třídách

Soubor `srs_env_model/src/but_server/octonode.cpp`:

- void octomap::EMOcTree::insertColoredScan(const typePointCloud& coloredScan, const octomap::point3d& sensor_origin, double maxrange, bool pruning, bool lazy_eval)
- octomap::EModelTreeNode* octomap::EMOcTree::integrateNodeColor(const OcTreeKey& key, const unsigned char& r, const unsigned char& g, const unsigned char& b, const unsigned char& a)
- octomap::EModelTreeNode* octomap::EMOcTree::setNodeColor(const OcTreeKey& key, const unsigned char& r, const unsigned char& g, const unsigned char& b, const unsigned char& a)
- void octomap::EModelTreeNode::setAverageChildColor()

- void octomap::EModelTreeNode::expandNode()
- bool octomap::EModelTreeNode::pruneNode()
- void octomap::EModelTreeNode::updateColorChildren()

Soubor `srs_env_model/include/but_server/octonode.h`:

- EModelTreeNode* setNodeColor(const float& x, const float& y, const float& z, const unsigned char& r, const unsigned char& g, const unsigned char& b, const unsigned char& a)
- EModelTreeNode* averageNodeColor(const float& x, const float& y, const float& z, const unsigned char& r, const unsigned char& g, const unsigned char& b, const unsigned char& a)
- EModelTreeNode* integrateNodeColor(const float& x, const float& y, const float& z, const unsigned char& r, const unsigned char& g, const unsigned char& b, const unsigned char& a)

Upravené existující metody

Soubor `but_srs/srs_ui_but/src/but_display/init.cpp`:

- void rvizPluginInit(rviz::TypeRegistry* reg)

  Přidán řádek pro registraci displeje `CButCamDisplay`

Soubor `srs_env_model/src/but_server/plugins/OctoMapPlugin.cpp`

- void srs::COctoMapPlugin::insertScan(const tf::Point& sensorOriginTf, const tPointCloud& ground, const tPointCloud& nonground)

  Změněno volání původní metody oktomapy z `insertScan` na volání nové metody `insertColoredScan`

Soubor `srs_env_model/src/but_server/plugins/PointCloudPlugin.cpp`

- void srs::CPointCloudPlugin::insertCloudCallback(const tIncommingPointCloud::ConstPtr& cloud)

  Upraveno vkládání nového point cloudu a jeho reprezentace pro uchovávání barvy jednotlivých bodů

B.0.2 Návod ke spuštění

Ke spuštění programové části je potřeba mít nainstalovaný ROS, programy jsou otestovány na verzi *Electric*. Instalace je relativně snadná, avšak pouze na systému Ubuntu 10.10 nebo 11.04.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu natty main" > \\
            /etc/apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install ros-electric-desktop-full
```

Dále je nutné mít správně nastavené proměnné prostředí.

```
source /opt/ros/electric/setup.bash
export ROS_ROOT=/opt/ros/electric/ros
export PATH=$ROS_ROOT/bin:$PATH
export PYTHONPATH=$ROS_ROOT/core/roslib/src:$PYTHONPATH
export ROBOT=cob3-3
export ROBOT_ENV=ipa-kitchen
```

Pro spuštění simulace Care-O-bota je nutné doinstalovat potřebné balíky.

```
sudo apt-get install ros-electric-care-o-bot
```

Některé součásti balíků SRS projektu nejspíš budou potřebovat nejnovější verze závislostí, stáhnutelné z githubu

```
sudo apt-get install git-core curl
wget https://github.com/ipa320/setup/raw/master/create_overlay.sh \
-N --no-check-certificate
chmod 755 create_overlay.sh

./create_overlay.sh cob_extern
./create_overlay.sh cob_common
./create_overlay.sh schunk_modular_robotics
./create_overlay.sh cob_driver
./create_overlay.sh cob_robots
./create_overlay.sh cob_environments
./create_overlay.sh cob_command_tools
./create_overlay.sh cob_simulation
./create_overlay.sh cob_navigation
```

Je nutné nastavit cestu k těmto staženým balíčkům a sloučit je s oficiální verzí.

```
export ROS_PACKAGE_PATH=~/git/care-o-bot:$ROS_PACKAGE_PATH
wget https://github.com/ipa320/setup/raw/master/githelper \
-N --no-check-certificate
chmod 755 githelper
./githelper merge
```

Když jsou nainstalovány všechny závislosti, je nutné, aby zdrojové soubory práce byly viditelné pro systém ROS. K tomu je určena systémová proměnná $ROS\_PACKAGE\_PATH$, v níž musí být nastavená cesta k novým zdrojovým souborům. Tyto soubory se nacházejí buď na přiloženém CD v adresáři `src/`. V poslední fázi je třeba již pouze zkompilovat dané soubory.

```
rosmake cob_bringup_sim
rosmake srs_ui_but
rosmake srs_env_model
```

V příkazech výše je `cob_bringup_sim` balíček, obstarávající kompletní spuštění simulace care-O-bota. `srs_ui_but` je balík obsahující variantu vizualizace s barevným polygonem zobrazeným před point cloudem. Balík `srs_env_model` je kompletní aktuální verze environmentálního modelu vytvořeného v rámci projektu SRS, který byl upravován v této práci pro vizualizaci obarveného point cloudu.

Je-li vše korektně přeloženo, obě vizualizace by měly být spustitelné každá jediným příkazem. Pro zobrazení první varianty stačí zadat:

```
roslaunch srs_ui_but data_fusion_test.launch
```

Zobrazení pohledového tělesa a barevného polygonu je možné dynamicky zapínat a vypínat v levém panelu aktivních displejů. Nastavitelné parametry lze rovněž měnit v panelu *properties* displeje `CButCamDisplay`.

Zobrazení barevného environmentálního modelu, tedy druhé varianty vizualizace je spouštěno příkazem:

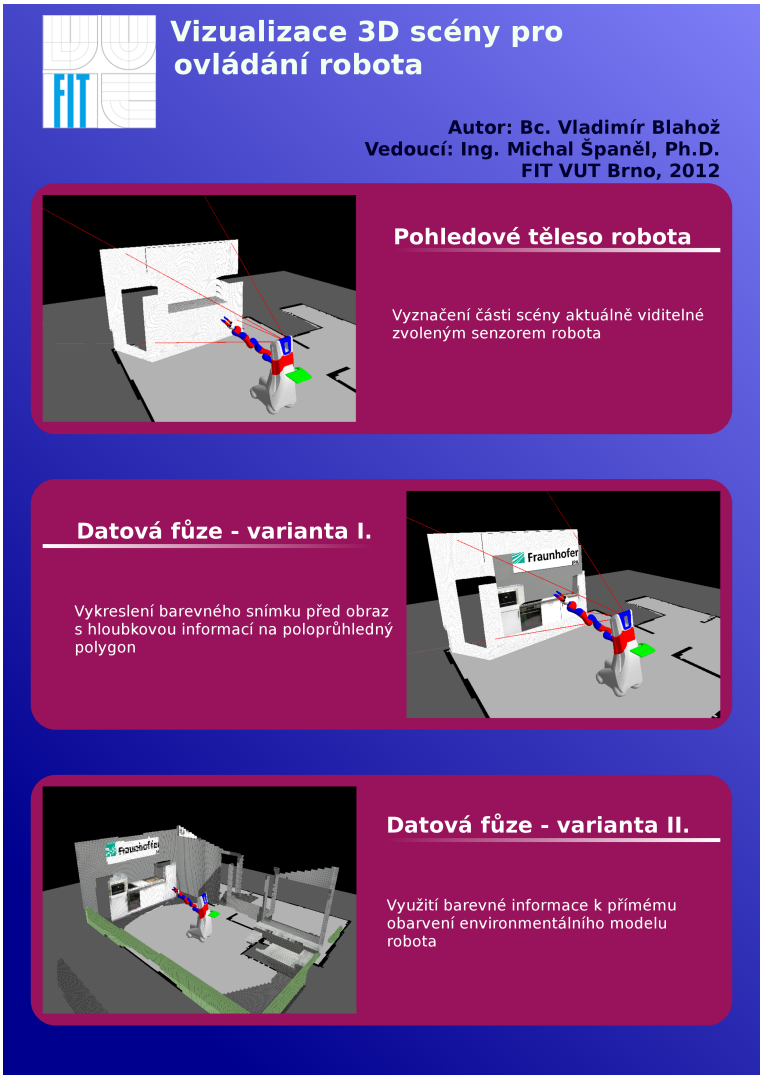
```
roslaunch srs_env_model colored_dynmodel.launch
```

V této variantě lze měnit parametry zobrazení point cloudu reprezentující environmentální model v panelu *properties* displeje *PointCloud2*.

Pohybovat s robotem lze u obou variant v perspektivě *Interact* (tlačítko poh hlavním menu prostředí Rviz, nebo klávesa `i`) taháním myší za části interaktivního markeru pod robotem.

Příloha C

Plakát



Obrázek C.1: Náhled plakátu prezentujícího diplomovou práci